PATH OPTIMIZATION FOR THE RESOURCE-CONSTRAINED SEARCHER

H. Sato and J.O. Royset
Operations Research Department, Naval Postgraduate School
Monterey, California, USA

Abstract. We formulate and solve a discrete-time path-optimization problem where a single searcher, operating in a discretized 3-dimensional airspace, looks for a moving target in a finite set of cells. The searcher is constrained by maximum limits on the consumption of several resources such as time, fuel, and risk along any path. We develop a specialized branch-and-bound algorithm for this problem that utilizes several network reduction procedures as well as a new bounding technique based on Lagrangian relaxation and network expansion. The resulting algorithm outperforms a state-of-the-art algorithm for solving time-constrained problems and also is the first algorithm to solve multi-constrained problems.

1 Introduction

We consider a discrete-time path-optimization problem where a single searcher moves through a discretized 3-dimensional airspace to find a moving target operating in a finite set of cells on the ground. The searcher is subject to constraints on path continuity, path endpoint, flight time, risk exposure, fuel consumption, and possibly other factors. We refer to this path-optimization problem as the resource-constrained search problem (RCSP). The objective of RCSP is to maximize the probability of detecting the target.

RCSP arises in military search, surveillance, and reconnaissance operations with patrol aircraft and unmanned aerial vehicles (UAVs) where physical and operational constraints limit the probability of detection. The resulting optimization problem is quite challenging. In fact, the path- and time-constrained search problem with a stationary target is NP-

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2008		**00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Path Optimization for the Resource-Constrained Searcher	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Operations Research Department,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We formulate and solve a discrete-time path-optimization problem where a single searcher, operating in a discretized 3-dimensional airspace, looks for a moving target in a finite set of cells. The searcher is constrained by maximum limits on the consumption of several resources such as time, fuel, and risk along any path. We develop a special- ized branch-and-bound algorithm for this problem that utilizes several network reduction procedures as well as a new bounding technique based on Lagrangian relaxation and net- work expansion. The resulting algorithm outperforms a state-of-the-art algorithm for solving time-constrained problems and also is the first algorithm to solve multi-constrained problems.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**48**	

complete [22]. Thus RCSP, which may involve a moving target and additional constraints, is at least as difficult.

Search for a moving target in discrete time and in an environment consisting of a finite set of cells was first analyzed in [21, 23]. These studies as well as more recent ones [11, 18, 8, 25, 17] focus on the development of specialized branch-and-bound algorithms for finding an optimal path for the searcher. In these algorithms, a path is a sequence of cells that the searcher will visit and branching corresponds to extending a subpath by one more cell. Bounds on the optimal value of the problem are obtained by replacing the probability of detection with, effectively, the expected number of detections [8, 25, 17]. Bounds are also obtained by assuming that the searcher can divide its effort among multiple cells each time period [11].

Existing studies only consider the path- and time-constrained problem where path continuity and a search-time limit are enforced. In addition, these studies limit the searcher to search on the ground or from a constant altitude. In military search, surveillance, and reconnaissance operations over land, however, factors such as risk and fuel consumption become independent elements of concern for planners [20]. Risk to the searcher arises from exposure to threats on the ground such as small arms fire, anti-aircraft artillery, and surface-to-air missiles. Risk of pilot error (e.g., for a low-flying helicopter [15]) and mechanical failure (e.g., for small, unreliable UAVs [16]) may also be significant. Fuel consumption ceases to be proportional to the search duration due to variation in the searcher's speed and/or altitude as well as varying weather conditions. During search over land, terrain features require altitude changes. The altitude is also varied to balance the searcher's risk with image quality. We note that image quality is of particular concern for small UAVs operating with low- to moderate-quality sensors. Our study is primarily motivated by military operations. However, civilian search and surveillance operations over land may face many of the same factors with the exception of ground fire.

Another line of research aims at finding a resource-constrained shortest path in a network. Such problems arise in minimum-risk routing of military aircraft subject to fuel [19, 26]

and fuel and time [6] constraints. These problems have similar constraints to the ones in RCSP, but deal with linear and time-invariant objective functions. In contrast, RCSP has a nonlinear, time-variant objective function where the probability of detection during the current time period depends on the subpath used to reach the searcher's current position (see Section 2). In the context of minimum-risk routing, researchers have only recently considered resource-constrained shortest-path problems with nonlinear objective functions, and then only for special cases of path-dependent risks [15].

In this paper, we formulate RCSP by generalizing existing models to the case of multiple resource constraints and an altitude-dependent searcher with glimpse detection probability depending on not only the searcher's current location but also its previous location. We combine the methodology for solving the path- and time-constrained search problem with the line of research on the resource-constrained shortest-path problem. Specifically, we merge the algorithms in [17] and [6], and develop a specialized branch-and-bound algorithm for RCSP. The resulting algorithm utilizes a new bounding technique, applies several network reduction procedures, exhibits promising behavior in computational tests, and appears to be easier to implement efficiently than the algorithm in [17].

The remainder of the paper is outlined as follows. The next section formulates RCSP. Section 3 describes an existing branch-and-bound algorithm and develops several enhancements. Section 4 extends the algorithm of Section 3 to the case with multiple resource constraints. Section 5 presents a case study of RCSP with time, risk, and fuel resource constraints. The paper ends with summary and conclusions in Section 6.

2 Resource-Constrained Search Problem

The area of interest is discretized into a finite set of cells $\mathcal{C} = \{1, \ldots, C\}$ (see Figure 1) and the time horizon is discretized into a finite set of time periods $\mathcal{T} = \{1, 2, ..., T\}$. A target occupies one cell each time period and moves among cells according to a Markov process with known transition matrix Γ. Let $p(\cdot, t) = [p(1, t), p(2, t), \ldots, p(C, t)]$, where $p(c, t)$ is the probability that the target is in cell $c \in \mathcal{C}$ at the beginning of time period $t \in \mathcal{T}$ and the

target has not been detected before t. We refer to $p(\cdot, t)$ as the undetected target distribution. The initial target distribution $p(\cdot, 1)$ is known.

The searcher moves through a designated airspace over the area of interest with the goal to find the moving target on the ground. The airspace over each cell $c \in \mathcal{C}$ is vertically discretized into a set of altitudes $\mathcal{H} = \{1, 2, \ldots, H\}$ (see Figure 2). For any $c \in \mathcal{C}$ and $h \in \mathcal{H}$, we refer to the cell-altitude pair $\langle c, h \rangle$ as a waypoint where the searcher can loiter and carry out search of cell c. We model the designated airspace by a directed network $(\mathcal{V}, \mathcal{E})$, with set of vertices $\mathcal{V}$ and set of directed edges $\mathcal{E}$, in which vertices $v = \langle c, h \rangle \in \mathcal{V}$ represent waypoints and directed edges $e = (v, v') \in \mathcal{E}$ represent transition between waypoints $v, v' \in \mathcal{V}$. The searcher can only transit between two waypoints that are physically located adjacent to each other. Let $\mathcal{F}(v) \subset \mathcal{V}$ be the set of vertices that are adjacent to $v \in \mathcal{V}$. We refer to $\mathcal{F}(v)$ as the forward star of vertex v. We adopt the convention that $v \in \mathcal{F}(v)$ for all $v \in \mathcal{V}$. Then, the set of edges $\mathcal{E} = \{(v, v') \in \mathcal{V} \times \mathcal{V} |\ v' \in \mathcal{F}(v)\}$.

During each time period $t \in \mathcal{T}$, the searcher is at a particular vertex (waypoint). We assume there is no transit time between waypoints. Hence, $(v, v') \in \mathcal{E}$ simply represents search at waypoint v followed by search at waypoint v' in the next time period. The situation with nonzero transit time between waypoints can be modeled, at least approximately, by introducing artificial vertices. (We refer to [17] for a comprehensive study of nonzero transit times.) We note that the edge $(v, v) \in \mathcal{E}$ represents searching at waypoint v for two consecutive time periods.

Let $\phi : \mathcal{V} \to \mathcal{C}$ be the function that specifies the cell over which a vertex is located, i.e., cell $\phi(v)$ is searched from vertex v. We denote the searcher's vertex (waypoint) prior to time period 1 by $v_0 \in \mathcal{V}$. This starting vertex could be a designated entry waypoint into the area of interest. We also define $\hat{\mathcal{V}} \subset \mathcal{V}$ to be a set of possible destination vertices for the searcher (e.g., $\hat{\mathcal{V}} = \{v_0\}$ if the searcher is required to return to the starting vertex or $\hat{\mathcal{V}} = \mathcal{V}$ if the search can end anywhere).

For any $k \in \mathcal{T}$ and $v_l \in \mathcal{V}$, $l = 0, 1, 2, ..., k$, such that $(v_{l-1}, v_l) \in \mathcal{E}$ for all $l = 1, 2, ..., k$, let the sequence $\{v_l\}_{l=0}^{k}$ denote a directed v_0-v_k subpath. If $v_k \in \hat{\mathcal{V}}$, then the directed v_0-v_k

subpath is a directed v_0-v_k path. When the meaning is clear from the context, we refer to a directed v_0-v_k (sub)path as a (sub)path. In this notation, the searcher flies from v_0 to some $v_k \in \hat{\mathcal{V}}$ along a directed v_0-v_k path. The searcher occupies only one vertex $v \in \mathcal{V}$ each time period, and stays at the same vertex or moves to another vertex in $\mathcal{F}(v)$ for the next time period.

We adopt the following target detection model. If the target is in cell $c \in \mathcal{C}$ during time period $t \in \mathcal{T}$ and the searcher is at the same time at waypoint $v' \in \mathcal{V}$ above cell c, i.e., $\phi(v') = c$, then detection occurs with a glimpse detection probability $g(v, v', t)$, where $v \in \mathcal{V}$ is the searcher's waypoint during time period $t-1$. Hence, the glimpse detection probability during time period t depends on the previous and current waypoints for the searcher. This is a generalization of earlier models where the glimpse detection probability depends only on v' and t [25, 17]. Our model accounts for the fact that moving from some waypoint to a new waypoint may result in a lower glimpse detection probability than if the searcher already was loitering at the latter waypoint. This effect occurs if refocusing the sensor and becoming familiar with a new cell have a significant detrimental effect on the glimpse detection probability. In general, change of waypoint, especially change of altitude and frequent, irregular change of direction, may distract from the search. This generalization also allows us to indirectly account for small transit times (much less than the length of a time period) between waypoints without adopting a fine time discretization with resulting high computational cost. In this notation, the probability of detection at waypoint v' during time period t, given search at waypoint v during time period $t-1$, and no prior detections becomes $p(\phi(v'), t)g(v, v', t)$.

The glimpse detection probability may also depend on the searcher's speed. To account for this situation, edges can trivially be duplicated as in [6] to represent search at different speeds in cases where speed influences the searcher's effectiveness significantly. For the sake of simplicity, however, we do not introduce notation to handle that situation.

Since $p(\cdot, t)$ is the undetected target distribution at the beginning of time period t, it depends on searches up to time period $t-1$. Specifically, if $p(\cdot, t) = [p(1, t), \ldots, p(c', t), \ldots, p(C, t)]$

and cell c' is searched from waypoint v' during time period t, the undetected target distribution at the beginning of the next time period $t+1$ is

$$p(\cdot, t+1) = [p(1,t), .., p(c'-1,t), p(c',t)(1-g(v,v',t)), p(c'+1,t), .., p(C,t)]\Gamma, \quad (1)$$

where v is the searcher's vertex during time period $t-1$. Thus, the probability of detection along a path $\mathcal{P} = \{v_l\}_{l=0}^{k}$, denoted $q(\mathcal{P})$, is defined as

$$q(\mathcal{P}) = \sum_{t=1}^{k} p(\phi(v_t), t) g(v_{t-1}, v_t, t). \quad (2)$$

Let $\mathcal{I} = \{1, 2, ..., I\}$ be a set of resources and $f_i(v, v', t)$ be the amount of resource $i \in \mathcal{I}$ consumed by the searcher at vertex v' during time period t, given search at vertex v during time period $t-1$. Resources may represent physical commodities such as fuel and ordnance that are depleted during the search as well as abstract factors such as notions of risk exposure during the search. In contrast to search by manned aircraft where significant risks are usually avoided, planners accept higher risks for UAV search missions and would like to balance risk with other factors during the planning process. We discuss the calculation of risk in Section 5. The total "consumption" of resource $i \in \mathcal{I}$ along the path $\mathcal{P}$ is

$$r_i(\mathcal{P}) = \sum_{t=1}^{k} f_i(v_{t-1}, v_t, t). \quad (3)$$

The searcher cannot consume more than $\hat{r}_i$ of resource $i \in \mathcal{I}$ along a path. Hence, RCSP is the problem to find a directed v_0-v_k path $\mathcal{P} = \{v_l\}_{l=0}^{k}$, with $v_k \in \hat{\mathcal{V}}$, $k \in \mathcal{T}$, that maximizes $q(\mathcal{P})$ subject to the constraints

$$r_i(\mathcal{P}) \leq \hat{r}_i, i \in \mathcal{I}. \quad (4)$$

We refer to (4) as side constraints. In Section 5, we examine a case study with two time-invariant resources: risk and fuel. The next section deals with the "unconstrained" problem with no side constraints, while Sections 4 and 5 address the full RCSP.

3 Branch-and-Bound Algorithm for Time-Constrained Problem

We refer to the problem of maximizing (2) without the side constraints (4) as the time-constrained search problem (TCSP). Several branch-and-bound algorithms for the solution of

TCSP have been developed [21, 23, 11, 18, 25, 8, 17]. These algorithms implicitly enumerate all searcher paths that cannot be proven, by means of a bound, to be nonimproving. The next subsection presents the algorithm in [17], which appears to be the fastest in the literature for TCSP under a broad range of conditions; see [17] for an empirical study. The subsequent subsection presents four modifications of that algorithm.

In this section we do not consider side constraints and can assume without loss of generality that an optimal path consists of $T+1$ vertices. To simplify the notation, we also assume in this section that there is no end-point restriction, i.e., $\hat{\mathcal{V}} = \mathcal{V}$. We find identical assumptions in [25, 17]. Initially, we assume that the glimpse detection probability $g(v, v', t)$ is independent of v and write $g(v', t)$. However, we relax that assumption later in this section.

3.1 Existing Algorithm

For completeness and ease of reference later, we outline the algorithm in [17]. Given a subpath $\{v_l\}_{l=0}^{t-1}$, $t \in \mathcal{T}$, let $\mathcal{K}(t)$ be the set of triplets of the form $(v_t, t, \bar{q}(v_t, t))$ representing extensions of $\{v_l\}_{l=0}^{t-1}$ yet to be explored. The first element v_t refers to the next vertex to visit, the second element t is the time period[1] to visit the vertex v_t, and the third element $\bar{q}(v_t, t)$ is an upper bound on the probability of detection along any path that starts with the subpath $\{v_l\}_{l=0}^{t}$. The upper bound $\bar{q}(v_t, t)$ consists of three parts. Let $d_t(v_t, t)$ be an upper bound on the probability of detection during time periods $t+1, t+2, ..., T$, given that the searcher starts at v_t at time t, and no detection occurs along the subpath $\{v_l\}_{l=0}^{t}$. The two other parts are the probability of detection on the subpath $\{v_l\}_{l=0}^{t-1}$ and the probability of detection during t. Hence,

$$\bar{q}(v_t, t) = q(\{v_l\}_{l=0}^{t-1}) + p(\phi(v_t), t)g(v_t, t) + d_t(v_t, t). \tag{5}$$

We also let $\hat{q}$ denote the largest detection probability found so far among all the examined paths. In this notation, the algorithm in [17] takes the following form.

Algorithm 1.

[1] This information is currently redundant but the notation is convenient in later generalizations.

Step 0. Set $t = 0, \mathcal{K}(t) = \{(v_0, 0, \infty)\}$, and $\hat{q} = 0$.

Step 1. If $\mathcal{K}(t)$ is empty, replace t by $t - 1$. Else, go to Step 3.

Step 2. If $t = 0$, stop: the last saved path is optimal and $\hat{q}$ is its probability of detection. Else, go to Step 1.

Step 3. Remove from $\mathcal{K}(t)$ the triplet $(v_t, t, \bar{q}(v_t, t))$ with the largest bound $\bar{q}(v_t, t)$.

Step 4. If $\bar{q}(v_t, t) \leq \hat{q}$, go to Step 1. (Current subpath is fathomed.)

Step 5. If $t < T$, then for each vertex $v \in \mathcal{F}(v_t)$, calculate a bound $d_{t+1}(v, t+1)$ as well as $\bar{q}(v, t+1)$, see (5), and add $(v, t+1, \bar{q}(v, t+1))$ to $\mathcal{K}(t+1)$. Replace t by $t+1$ and go to Step 3. Else, let $\hat{q} = \bar{q}(v_t, t)$ and save the incumbent path $\{v_l\}_{l=0}^{T}$, and go to Step 1.

Clearly, a tight bound $d_t(v_t, t)$ will reduce the number of branching attempts in Algorithm 1. As examined in [24, 25, 17], there is a fundamental trade-off between the effort needed to compute a bound and its tightness. From these studies, it appears that the bounding technique in [17] compares favorably in most situations. We describe that bounding technique in the rest of this subsection.

Consider a subpath $\{v_l\}_{l=0}^{t}$, $t \in \mathcal{T}$, and let $p_g(\cdot, t)$ be the undetected target distribution after search along $\{v_l\}_{l=0}^{t}$, i.e.,

$$p_g(\cdot, t) = [p(1, t), ..., p(\phi(v_t) - 1, t), p(\phi(v_t), t)(1 - g(v_t, t)), p(\phi(v_t) + 1, t), ..., p(C, t)]. \quad (6)$$

We use subscript g to indicate that $p_g(\cdot, t)$ is obtained from $p(\cdot, t)$ by applying the glimpse detection probability corresponding to the last vertex in the "current" subpath $\{v_l\}_{l=0}^{t}$. For any integer $s > t$, $s \in \mathcal{T}$, we also define

$$p_\Gamma(\cdot, s; t) = p_g(\cdot, t)\Gamma^{s-t}. \quad (7)$$

As seen, $p_\Gamma(c, s; t)$ is the probability that the target is in cell c at time period $s > t$ and there was no detection during search along the subpath $\{v_l\}_{l=0}^{t}$. Hence, target distribution

$p_\Gamma(\cdot, s; t)$ at time period $s > t$ ignores the effect of search after time period t. If the subpath is $\{v_0\}$, i.e., $t = 0$, we define for notational convenience

$$p_\Gamma(\cdot, s; 0) = p(\cdot, 1)\Gamma^{s-1}, \tag{8}$$

for any $s > 0$, $s \in \mathcal{T}$. Moreover, we define $p_\Gamma(c, t; t) = 0$ for all $c \in \mathcal{C}$ and $t = 0, 1, ..., T$.

Now, we construct a time-expanded graph from the network $(\mathcal{V}, \mathcal{E})$ as follows (see Figure 3). Each vertex $v \in \mathcal{V}$ is duplicated T times to define the nodes $\langle v, s\rangle$, $s \in \mathcal{T}$. Let $\mathcal{N}$ be the set of all such nodes as well as the nodes $n_0 = \langle v_0, 0\rangle$ and $\hat{n} = \langle \hat{v}, T + 1\rangle$ representing the searcher's prior position and final position, respectively. Here, $\hat{v}$ is an artificial terminal vertex. Two nodes $n = \langle v, s - 1\rangle$ and $n' = \langle v', s\rangle$, $v, v' \in \mathcal{V}$ and $s = 2, 3, ..., T$, are connected with an arc (n, n') if and only if $(v, v') \in \mathcal{E}$. Moreover, the node $n_0 = \langle v_0, 0\rangle$ is connected with an arc to a node $n' = \langle v', 1\rangle$, $v' \in \mathcal{V}$, if and only if $(v_0, v') \in \mathcal{E}$; and every node $n = \langle v, T\rangle$, $v \in \mathcal{V}$ is connected with an arc to $\hat{n}$. Let $\mathcal{A}$ be the set of all arcs. For any integer $k \leq T + 1$ and nodes $n_l = \langle v_l, l\rangle \in \mathcal{N}, l = 0, 1, ..., k$, such that $(n_{l-1}, n_l) \in \mathcal{A}$ for all $l = 1, 2, ..., k$, we let the sequence $\{n_l\}_{l=0}^k$ denote a subpath in the time-expanded graph $(\mathcal{N}, \mathcal{A})$.

For some $t \in \{0, 1, ..., T - 1\}$, suppose that a subpath $\{v_l\}_{l=0}^t$ in the original graph $(\mathcal{V}, \mathcal{E})$ is given. Then, we endow each arc $(n, n') = (\langle v, s\rangle, \langle v', s + 1\rangle) \in \mathcal{A}$, $s = t, t + 1, ..., T - 1$, in the time-expanded graph $(\mathcal{N}, \mathcal{A})$ with a "reward"

$$c_{n,n'} = [p_\Gamma(\phi(v'), s + 1; t) - p_\Gamma(\phi(v), s; t)g(v, s)\Gamma(v, v')]g(v', s + 1), \tag{9}$$

where $\Gamma(v, v')$ is the $\phi(v)$-$\phi(v')$ element of the Markov transition matrix Γ. We set $c_{n,\hat{n}} = 0$ for all $(n, \hat{n}) \in \mathcal{A}$. We observe that multiplied out the first term $p_\Gamma(\phi(v'), s + 1; t)g(v', s + 1)$ in (9) is effectively the expected number of detections during time period $s + 1$, which gives rise to the so-called mean bound [18], and the second term in (9) improves the bound by accounting for the effect of search during time period s [17]. We refer to $(\mathcal{N}, \mathcal{A})$ with arc rewards given by (9) as the time-expanded network.

In [17], it is shown that given the subpath $\{v_l\}_{l=0}^t$, the optimal value of the longest-path problem in the time-expanded network from node $\langle v_t, t\rangle$ to node $\langle \hat{v}, T+1\rangle$, using the rewards in (9) as "arc length," provides an upper bound for Algorithm 1. Specifically, this optimal

value is an upper bound on the probability of detection during time periods $t+1, t+2, ..., T$, given that the searcher starts at v_t at time t, and no prior detections occurred along the subpath $\{v_l\}_{l=0}^{t}$. We denote this bound by $d_t(v_t, t)$ and refer to it as the dynamic bound as it needs to be recomputed every time the *current* subpath is extended.

Since the time-expanded network is acyclic, the longest-path problem can be solved in polynomial calculation time with a standard shortest-path algorithm ([1], pages 77-79). We observe that to compute $d_t(v_t, t)$ given the subpath $\{v_l\}_{l=0}^{t}$, it is only necessary to generate the part of the graph $(\mathcal{N}, \mathcal{A})$, and the corresponding arc rewards, "after" time t and within reach from node $\langle v_t, t\rangle$, since the longest path starts at node $\langle v_t, t\rangle$. Hence, in Step 5 of Algorithm 1, it suffices to generate the time-expanded network "after" time t.

3.2 Algorithmic Modifications for TCSP

In this subsection, we propose and examine four modifications of Algorithm 1. The first modification extends the bound in [17] to the case with glimpse detection probability depending on the previous vertex, i.e., $g(v, v', t)$. The second modification simplifies the bound calculation substantially at the expense of a weaker bound. The third modification improves the weaker bound. The fourth modification takes advantage of a special, but frequently occurring, initial target distribution.

3.2.1 Bound for Edge-Dependent Glimpse Detection Probability

In previous studies (see, e.g., [25, 17]), the glimpse detection probability is assumed to depend on the searcher's current waypoint (vertex) and time. As we argue in Section 2, this is somewhat restrictive. Fortunately, we can easily extend this to the case where the glimpse detection probability also depends on the searcher's previous waypoint. The only modification that is required is to redefine the arc reward $c_{n,n'}$ in (9). However, a straightforward replacement of $g(v, s)$ by $g(v'', v, s)$ in (9), where v'' is the vertex prior to v, would ruin the longest-path structure of the bound calculation problem: $c_{n,n'}$ would no longer only depend on the head and tail of the arc (n, n'). Hence, it is necessary to use the smallest glimpse detection probability $\min_{v'' \in \mathcal{R}(v)} g(v'', v, s)$ to eliminate the dependence on the vertex prior

to v. Consequently, we now endow each arc $(n, n') = (\langle v, s\rangle, \langle v', s+1\rangle) \in \mathcal{A}$ with the reward

$$c_{n,n'} = \left[p_\Gamma(\phi(v'), s+1; t) - p_\Gamma(\phi(v), s; t) \left(\min_{v'' \in \mathcal{R}(v)} g(v'', v, s) \right) \Gamma(v, v') \right] g(v, v', s+1), \quad (10)$$

where $\mathcal{R}(v) \subset \mathcal{V}$ is the reverse star of v, i.e., $\mathcal{R}(v) = \{v'' \in \mathcal{V} \mid (v'', v) \in \mathcal{E}\}$. With this reward, the bound calculation remains a longest-path problem in an acyclic graph and it can be shown using the same arguments as in [17] that the dynamic bound is valid.

3.2.2 Static Bound

Algorithm 1 requires one longest-path calculation in the time-expanded network for each vertex in the forward star of the current vertex to compute the required bounds $d_t(v_t, t)$ (see Step 5). The approach of Algorithm 1 with dynamic bound follows the traditional approach of branch-and-bound algorithms where the bound is reoptimized before each branching. In the present case, the reoptimization corresponds to the longest-path calculation and requires computing the arc rewards $c_{n,n'}$, see (10). These calculation can be time-consuming as they typically involve a moderately large Markov transition matrix Γ and associated matrix multiplication. We propose to use a static bound instead of the dynamic bound proposed in [17] and described in Subsection 3.1. As shown below, all the necessary static bounds are computed prior to any branching and are *not* recomputed later.

The dynamic bound $d_t(v_t, t)$ (see Subsection 3.1) uses information about search along a current subpath $\{v_l\}_{l=0}^t$. However, the bound remains valid if the effect of search along the current subpath is ignored. This follows from the same arguments as in the proof of the validity of $d_t(v_t, t)$, see [17]. We denote the new bound $d_0(v_t, t)$, where the subscript 0 indicates that the trivial subpath $\{v_0\}$ is used in (10) with $t = 0$ instead of the subpath $\{v_l\}_{l=0}^t$, which $d_t(v_t, t)$ utilizes. Hence, the only difference between $d_t(v_t, t)$ and $d_0(v_t, t)$ is that $p_\Gamma(\cdot, \cdot; t)$ is replaced by $p_\Gamma(\cdot, \cdot; 0)$ in (10). However, this difference makes the bound $d_0(v_t, t)$ independent of the current subpath used to reach the vertex v_t. Hence, $d_0(v, t)$ can be computed in advance for all nodes $\langle v, t\rangle \in \mathcal{N}$, and dynamical computation of bounds is not required. Consequently, the arc rewards (10) and bounds are computed only once. We refer to $d_0(v, t)$ as the static bound. We observe that it is not necessary to carry out

a longest-path calculation from each node $\langle v, t\rangle \in \mathcal{N}$ to $\langle \hat{v}, T+1\rangle$ to obtain $d_0(v,t)$. It is more efficient to carry out the longest-path calculations backward from node $\langle \hat{v}, T+1\rangle$ to all nodes. This calculation simply amounts to applying once a shortest-path algorithm to the time-expanded network with arc lengths equal to the negative rewards.

In Step 5 of Algorithm 1, we now simply use $d_0(v_t, t)$ instead of $d_t(v_t, t)$. Thus, the modified algorithm does not require any longest-path calculation in Step 5. All bound calculations are done prior to Step 0. Clearly, the modified approach results in a weaker bound and the need for more branching attempts. However, the additional branching attempts may be compensated by shorter per-iteration computing times.

In order to examine the effect of the static bound $d_0(v_t, t)$, we examine the same numerical example as in [17]: An area of interest consists of 11 by 11 cells. The searcher operates only at one altitude and its moves are restricted to vertically and horizontally adjacent cells, excluding diagonal moves. The target remains in the current cell with a probability ρ or moves to one of the vertically or horizontally adjacent cells with probability $1-\rho$. The different moves are equally likely. The searcher departs cell 1 ($v_0 = 1$), where the cells are numbered from left to right and from top to bottom. Hence, cell 1 is the upper-left-corner cell. The target starts at the center cell, i.e., at cell 61. The time horizon $T = 17$.

We implemented Algorithm 1 with static bound using Microsoft Visual C++ 6.0 on a desktop computer with a 3.4 GHz Intel Pentium IV processor, 1.0 gigabytes of RAM, and the Microsoft Windows XP operating system. Table 1 shows, for a range of constant glimpse detection probabilities $g(v, v', t)$ and "stay probabilities" ρ, the run times (in seconds) and numbers of bounding attempts of Algorithm 1 as well as those reported in [17]. We especially consider the case of high glimpse detection probability ($g(v, v', t) = 0.99$), in which both static and dynamic bounds tend to be relatively weak. Column 3 of Table 1 presents the resulting run times for Algorithm 1 with static bound. The corresponding run times reported from [17] are found in column 5. (The case $g(v, v', t) = 0.99$ is not considered in [17].) Those reported numbers are achieved on a 2.6 GHz Opteron 152 processor computer using Matlab. Hence, a direct comparison between the run times in columns 3 and 5 is difficult. However, we find

reports of run times for other problem instances using a C++ implementation in [17]. A brief comment in [17] based on a single problem instance indicates that the Matlab implementation is 15.6 times slower than the C++ implementation. For the sake of comparison, we scale down the run times in column 5 of Table 1 with 1/15.6 to approximately account for the slower Matlab implementation. We also scale down the run times of column 5 by a factor of 2.6/3.4 to account for the slower computer used in [17]. The resulting scaled down run times are presented in column 6 of Table 1. We observe that the scaled down run times for Algorithm 1 with dynamic bound are somewhat faster than the corresponding run times with static bound. However, the static bound, which is weaker than the dynamic bound, remains fairly competitive, especially for more difficult problem instances.

We compare the strengths of the static and dynamic bounds by counting the number of branching attempts required in Algorithm 1. Columns 4 and 7 of Table 1 give the numbers of branching attempts for the static and dynamic bounds, respectively. The number of branching attempts for the static bound is, on average, 37 times larger than in the case of the dynamic bound. We observe that the greater number of bounding attempts is partially compensated for by avoiding dynamical reoptimization of the bound.

For a direct comparison between the static and dynamic bounds, we implement Algorithm 1 with dynamic bound in Microsoft Visual C++ 6.0. We made a significant effort to ensure that the implementation is efficient, including efficient handling of sparse matrices. Column 8 and 9 of Table 1 report the run times and the number of branching attempts for our implementation of Algorithm 1 with dynamic bound, respectively. We observe that our implementation results in identical numbers of branching attempts compared to the implementation in [17]. When comparing columns 6 and 8, we see that our implementation of the dynamic bound results in somewhat longer run times than the scaled times from [17]. However, the longer times in column 8 compared to column 6 can partially be due to an excessively aggressive scaling of run times going from column 5 to column 6.

While implementing the dynamic bound, we noted a significant challenge associated with efficient matrix multiplication and data handling. Since the dynamic bound $d_t(v, t)$ is

reoptimized a large number of times, it is paramount to carry out the related calculations efficiently. In comparison, it is rather trivial to implement a static bound. It is not critical to carry out the longest-path calculations highly efficiently in the static case as they are only done once.

We also observe that both static and dynamic bounds tend to be weaker when the target is near stationary (e.g., $\rho = 0.9$) and glimpse detection probability is large (e.g., $g(v, v', t) = 0.9$ or 0.99). For these problem instances, the implementation with dynamic bound is more sensitive to the change in data. In fact, comparing the instance ($g(v, v', t) = 0.9$ and $\rho = 0.9$) to the instance ($g(v, v', t) = 0.99$ and $\rho = 0.9$), we see that the run time and the number of branching attempts in the case of the dynamic bound become 1.29 and 1.48 times larger, respectively. On the other hand, the static bound is less sensitive, and its numbers become only 1.09 times larger. These numbers indicate that it becomes even less worthwhile to invest time in dynamic bound calculations when the bounds are relatively weak anyways.

3.2.3 Directional Static Bound

As seen from Table 1, the static bound is substantially weaker than the dynamic bound. We derive a stronger static bound motivated by the classical approach to handling turn-radius constraints in vehicle routing problems [4].

In the longest-path calculations for the static bound, the reward of arc $(\langle v, s\rangle, \langle v', s+1\rangle)$ is, effectively, the probability of detection at vertex v' during time period $s + 1$ and no detection at vertex v during time period s. Of course, this overestimates the probability of detection at vertex v' during time period $s + 1$ and no prior detections, as detection could occur prior to time period s. We strengthen the static bound if we redefine the arc reward to be the probability of detection at vertex v' during time period $s + 1$ and no detection at vertex v during time period s *and* no detection at the vertex visited during time period $s-1$. However, redefining the arc reward to depend not only on the arc's head and tail nodes, but also on a previous node ruins the longest-path structure of the bound-calculation problem.

A similar situation arises in vehicle routing problems for vehicles with turn-radius con-

straints or penalties. The classical approach to handle that situation is to duplicate each node a number of times equal to the number of nodes in the node's reverse star. An arc in the resulting "node-expanded" network then carries information about three nodes, not only two, and a desirable network structure of the problem can be maintained. Fortunately, it is practical to carry out such a node-expansion approach in the problems of interest in this paper because the number of nodes in the reverse star is typically quite moderate. Hence, we proceed along the stated lines and develop a node-and-time expanded network, in which the improved static bound can be calculated by solving a longest-path problem. We refer to this improved bound as the directional static bound.

For any $n' \in \mathcal{N}$, let $\mathcal{R}(n') \subset \mathcal{N}$ be the reverse star of n', i.e., $\mathcal{R}(n') = \{n \in \mathcal{N} | (n, n') \in \mathcal{A}\}$. Then, for any $n, n' \in \mathcal{N} \backslash \{\hat{n}\}$ such that $(n, n') \in \mathcal{A}$, we define an expanded node $\xi = \langle n, n' \rangle$. We do not expand the end node, so we set $\hat{\xi} = \hat{n}$. Let Ξ be the set of all expanded nodes. Two expanded nodes $\xi, \xi' \in \Xi$ are connected by an expanded arc (ξ, ξ') if $\xi = \langle n, n' \rangle$ and $\xi' = \langle n', n'' \rangle$. Let the set of all expanded arcs be Ω. The node-and-time expanded graph is illustrated in Figure 4.

We endow each expanded arc in the node-and-time expanded graph (Ξ, Ω) with a reward similar to (10). To derive the exact form of this reward, we need the following building blocks. For any $v, v' \in \mathcal{V}$ and $t \in \mathcal{T}$, let $M_t(v, v')$ be a C by C identity matrix with the $\phi(v')$-th diagonal element set equal to $1 - g(v, v', t)$. We also let $\Gamma(v')$ be the $\phi(v')$-th column of the Markov transition matrix Γ.

From (2) and the recursive application of (1), we see that the probability of detection along a path $\{v_l\}_{l=0}^T$ is given by

$$\begin{aligned} q(\{v_l\}_{l=0}^T) = {} & p(\phi(v_1), 1) g(v_0, v_1, 1) + \\ & p(\cdot, 1) M_1(v_0, v_1) \Gamma(v_2) g(v_1, v_2, 2) + \\ & p(\cdot, 1) M_1(v_0, v_1) \Gamma M_2(v_1, v_2) \Gamma(v_3) g(v_2, v_3, 3) + \\ & p(\cdot, 1) M_1(v_0, v_1) \Gamma M_2(v_1, v_2) \Gamma M_3(v_2, v_3) \Gamma(v_4) g(v_3, v_4, 4) + \\ & \vdots \end{aligned} \tag{11}$$

$$p(\cdot,1)M_1(v_0,v_1)\Gamma M_2(v_1,v_2)\Gamma M_3(v_2,v_3)\cdot\ldots\cdot\Gamma M_{T-1}(v_{T-2},v_{T-1})\Gamma(v_T)g(v_{T-1},v_T,T).$$

The expression (11) gives insight into a class of bounds on the probability of detection including the static bound $d_0(v_t,t)$. If we replace $M_t(\cdot,\cdot)$ by the identity matrix in (11), we find that

$$\begin{aligned} q(\{v_l\}_{l=0}^T) \leq\ & p(\phi(v_1),1)g(v_0,v_1,1) + \\ & p(\cdot,1)\Gamma(v_2)g(v_1,v_2,2) + \\ & p(\cdot,1)\Gamma\Gamma(v_3)g(v_2,v_3,3) + \\ & p(\cdot,1)\Gamma\Gamma\Gamma(v_4)g(v_3,v_4,4) + \\ & \vdots \\ & p(\cdot,1)\Gamma^{T-2}\Gamma(v_T)g(v_{T-1},v_T,T). \end{aligned} \tag{12}$$

In (12), the "reward" received during a time period is simply the expected number of detection during that time period and depends only on the current and previous vertices. Hence, it is possible to compute an upper bound on the optimal probability of detection by finding a path $\{v_l\}_{l=0}^T$ that maximizes the right-hand side in (12). This calculation amounts to a longest-path problem and is, in fact, the approach in [18]. (Note, however, that [18] assumes that the glimpse detection probability is independent of the previous vertex.)

If we replace each $M_t(\cdot,\cdot)$ by the identity matrix everywhere except the last matrix of each line in (11), we obtain

$$\begin{aligned} q(\{v_l\}_{l=0}^T) \leq\ & p(\phi(v_1),1)g(v_0,v_1,1) + \\ & p(\cdot,1)M_1(v_0,v_1)\Gamma(v_2)g(v_1,v_2,2) + \\ & p(\cdot,1)\Gamma M_2(v_1,v_2)\Gamma(v_3)g(v_2,v_3,3) + \\ & p(\cdot,1)\Gamma\Gamma M_3(v_2,v_3)\Gamma(v_4)g(v_3,v_4,4) + \\ & \vdots \\ & p(\cdot,1)\Gamma^{T-2}M_{T-1}(v_{T-2},v_{T-1})\Gamma(v_T)g(v_{T-1},v_T,T). \end{aligned} \tag{13}$$

Now, the reward received during each time period also depends on the searcher's position two time periods ago and the problem of finding a path that maximizes the right-hand side

is no longer a longest-path problem. However, the bound remains valid with the following minor modification, where the maximization of a matrix with a single element different from zero or one is simply the maximization of that element:

$$\begin{aligned} q(\{v_l\}_{l=0}^T) \leq\ & p(\phi(v_1),1)g(v_0,v_1,1) + \\ & p(\cdot,1)\left(\max_{v\in\mathcal{R}(v_1)} M_1(v,v_1)\right)\Gamma(v_2)g(v_1,v_2,2) + \\ & p(\cdot,1)\Gamma\left(\max_{v\in\mathcal{R}(v_2)} M_2(v,v_2)\right)\Gamma(v_3)g(v_2,v_3,3) + \\ & p(\cdot,1)\Gamma\Gamma\left(\max_{v\in\mathcal{R}(v_3)} M_3(v,v_3)\right)\Gamma(v_4)g(v_3,v_4,4) + \\ & \vdots \\ & p(\cdot,1)\Gamma^{T-2}\left(\max_{v\in\mathcal{R}(v_{T-1})} M_{T-1}(v,v_{T-1})\right)\Gamma(v_T)g(v_{T-1},v_T,T). \end{aligned} \tag{14}$$

After this modification, we see that the reward during each time period only depends on the current and previous vertices. Hence, again, it is possible to compute an upper bound on the optimal probability of detection by solving a longest-path problem. In fact, this is exactly the approach we described in Subsection 3.2.1 and it can be shown that the reward in the longest-path problem $c_{n,n'}$, see (10), can be deduced from (14). Specifically, when the current subpath in (10) is $\{v_0\}$, we have for arc $(n,n') = (\langle v,s\rangle, \langle v',s+1\rangle) \in \mathcal{A}$ that

$$c_{n,n'} = p(\cdot,1)\Gamma^{s-1}\left(\max_{v''\in\mathcal{R}(v)} M_s(v'',v)\right)\Gamma(v')g(v,v',s+1). \tag{15}$$

Using similar arguments, we define the directional static bound as follows. Clearly,

$$\begin{aligned} q(\{v_l\}_{l=0}^T) \leq\ & p(\phi(v_1),1)g(v_0,v_1,1) + \\ & p(\cdot,1)M_1(v_0,v_1)\Gamma(v_2)g(v_1,v_2,2) + \\ & p(\cdot,1)\left(\max_{v\in\mathcal{R}(v_1)} M_1(v,v_1)\right)\Gamma M_2(v_1,v_2)\Gamma(v_3)g(v_2,v_3,3) + \\ & p(\cdot,1)\Gamma\left(\max_{v\in\mathcal{R}(v_2)} M_2(v,v_2)\right)\Gamma M_3(v_2,v_3)\Gamma(v_4)g(v_3,v_4,4) + \\ & \vdots \\ & p(\cdot,1)\Gamma^{T-3}\left(\max_{v\in\mathcal{R}(v_{T-2})} M_{T-2}(v,v_{T-2})\right)\Gamma M_{T-1}(v_{T-2},v_{T-1})\Gamma(v_T)g(v_{T-1},v_T,T). \end{aligned} \tag{16}$$

Hence, we can compute an upper bound on the optimal probability of detection by finding a path $\{v_l\}_{l=0}^T$ that maximizes the right-hand side of (16). This calculation amounts to a longest-path problem in the node-and-time expanded graph (Ξ, Ω). The arc reward in this longest-path problem is deduced from (16). Specifically, an expanded arc $(\xi, \xi') = (\langle n'', n\rangle, \langle n, n'\rangle) \in \Omega$, with $n'' = \langle v'', s-1\rangle$, $n = \langle v, s\rangle$, and $n' = \langle v', s+1\rangle$, is endowed with the reward

$$c_{\xi,\xi'} = p(\cdot, 1)\Gamma^{s-2}\left(\max_{v'''\in\mathcal{R}(v'')} M_{s-1}(v''', v'')\right)\Gamma M_s(v'', v)\Gamma(v')g(v, v', s+1). \tag{17}$$

We refer to the node-and-time expanded graph (Ξ, Ω) with the arc rewards $c_{\xi,\xi'}$ from (17) as the node-and-time expanded network. Since the node-and-time expanded graph is acyclic, longest-path problems are solvable by standard shortest-path algorithms.

In view of the above discussion, we obtain the following result.

Proposition 1 *For any $v' \in \mathcal{V}$ and $t \in \mathcal{T}$, let*

(i) $d_0(v', t)$ be the value of the longest-path from node $\langle v', t\rangle$ to node $\hat{n}$ in the time-expanded graph $(\mathcal{N}, \mathcal{A})$ with arc rewards given by (15), and

(ii) $\delta_0(v, v', t)$ be the value of the longest-path from expanded node $\langle\langle v, t-1\rangle, \langle v'\ t\rangle\rangle$ to expanded node $\hat{\xi}$ in the node-and-time expanded graph (Ξ, Ω) with arc rewards given by (17).

Then, both $d_0(v', t)$ and $\delta_0(v, v', t)$ are upper bounds on the probability of detection during time periods $t+1, t+2, ..., T$ for any path $\{v_l\}_{l=0}^T$ with $v_{t-1} = v$ and $v_t = v'$. Moreover, $\delta_0(v, v', t) \leq d_0(v', t)$.

We refer to $\delta_0(v, v', t)$ as the directional static bound and see from Proposition 1 that it is at least as strong as the static bound. We demonstrate in an empirical study below that it may be substantially stronger.

Clearly, building the node-and-time expanded graph (Ξ, Ω), computing the associated rewards, and calculating the longest-paths take some computing time. However, the process

is only carried out once before the start of Algorithm 1 and the computed bounds are stored for later use. Hence, the time for computing the directional static bounds remains small compared to the overall run time of Algorithm 1. We conjecture that the use of the node-and-time expanded graph (Ξ, Ω) with dynamic reoptimization of bounds will not be efficient due to the small but significant effort required to build the node-and-time expanded graph, compute associated arc rewards, and carry out the longest-path calculations. We observe that this conjecture appears to be aligned with [17], which eludes to the inefficiency of a dynamic bound based on more than one-time-step look-behind.

In Table 2, we report computational results for Algorithm 1 with the directional static bound applied to the same problem instances as in Table 1. Columns 3 and 4 give run times and number of branching attempts, respectively, for Algorithm 1 using the directional static bound. We observe that, on average, the number of branching attempts has been reduced to 58.1% by using the directional static bound compared with the static bound. (Compare column 4 in Table 1 with column 4 in Table 2.) Similarly, the run times have been reduced to 58.2% by using the directional static bound. Since the reduction in branching attempts and run time are essentially identical, we conclude that the time for computing the directional static bound is small compared to the overall run time.

3.2.4 Network Reduction

In some practical situations, the searcher's and the target's initial positions are relatively far from each other. Hence, for a number of time periods the searcher will only examine cells where the target is guaranteed not to be located. In these initial moves, the searcher is simply positioning itself for the later search. This is the situation in the numerical examples of [17]. In this subsection, we derive a network reduction technique that utilizes this situation.

Consider the time-expanded graph $(\mathcal{N}, \mathcal{A})$. Suppose that the searcher flies, for the first s time periods, along a subpath $\{n_t\}_{t=0}^{s}$, with $n_t = \langle v_t, t \rangle$, such that $p(\phi(v_t), t) = 0$ for all $t < s$, and $p(\phi(v_s), s) > 0$. Then, the searcher cannot detect the target prior to time period s. We refer to the last node n_s of the subpath $\{n_t\}_{t=0}^{s}$ as a node-of-first-contact. It is

typically straightforward to determine a set of nodes-of-first-contact by applying a standard search algorithm ([1], pages 73-77) on the time-expanded network. In Figure 5, we illustrate a situation where the searcher can at the earliest detect the target during time period 4, and also note that the time period of all nodes-of-first-contact $\langle v_s, s\rangle \in \mathcal{N}$ is $s \geq 4$.

We observe that there might be several different subpaths $\{n_t\}_{t=0}^{s}$, $n_t = \langle v_t, t\rangle$, to reach a node-of-first-contact $\langle v_s, s\rangle \in \mathcal{N}$. However, the subpath used to reach $\langle v_s, s\rangle$ is of no or little importance. In fact, if $g(v_{s-1}, v_s, s)$ does not vary with the choice of v_{s-1}, all subpaths to $\langle v_s, s\rangle$ have probability of detection equal to $p(\phi(v_s), s)g(v_{s-1}, v_s, s)$, with $p(\phi(v_s), s) = p(\cdot, 1)\Gamma^{s-2}\Gamma(v_s)$. Thus, it is enough to find a subpath to each node-of-first-contact $\langle v_s, s\rangle$ using a standard search algorithm, connect initial node $\langle v_0, 0\rangle$ to $\langle v_s, s\rangle$ directly with a "jump" arc representing the move to $\langle v_s, s\rangle$, and ignore time periods $1, 2, ..., s-1$ during the branch-and-bound algorithm. Clearly, this procedure may reduce the amount of branching attempts significantly. We can generalize this to the case with $g(v_{s-1}, v_s, s)$ varying with respect to v_{s-1}, as discussed in Subsection 4.2.3.

If a lower bound $\hat{q}$ on the optimal probability of detection is available in advance of the network reduction procedure described above, it may not be necessary to consider all nodes-of-first-contact. Fortunately in TCSP, a lower bound is trivially obtained by computing the probability of detection along the path corresponding to the static bound $d_0(v_0, 0)$. Furthermore, for each node-of-first-contact $\langle v_s, s\rangle$, an upper bound on the probability of detection after time period s with no prior detections (e.g., the static bound $d_0(v_s, s)$) is available in advance of both branch-and-bound and network reduction procedures. The upper and lower bounds can be used to eliminate some nodes-of-first-contact that cannot lie on an optimal path. Specifically, if $p(\phi(v_s), s)g(v_{s-1}, v_s, s) + d_0(v_s, s) \leq \hat{q}$, we can eliminate $\langle v_s, s\rangle$ and all incoming and outgoing arcs. Thus, the amount of branching attempts may be reduced further.

In order to take advantage of this reduced network in the branch-and-bound algorithm, we construct a second algorithm (Algorithm 2) that generalizes the branch-and-bound mechanism of Algorithm 1. Before we describe the algorithm, we need to clarify some notation.

After the algorithm is presented, we describe the network reduction procedure in detail.

Given a subpath $\{n_l\}_{l=0}^{k}$, $n_l = \langle v_l, l\rangle$, $k \in \mathcal{T}$, let $\mathcal{K}(i)$, as before, be the set of triplets $(v_t, t, \bar{q}(v_t, t))$ representing extensions of $\{n_l\}_{l=0}^{k}$ yet to be explored. Now, the index i of $\mathcal{K}(i)$ refers to the depth from node $\langle v_0, 0\rangle$ to the next node $\langle v_t, t\rangle$ in the branch-and-bound tree. Since $\langle v_0, 0\rangle$ is directly connected to each node-of-first-contact $\langle v_s, s\rangle$, the corresponding depth in the branch-and-bound tree is 1. For reporting purposes, a subpath $\{n_l\}_{l=0}^{s}$ is stored for each node-of-first-contact $\langle v_s, s\rangle$.

Algorithm 2.

Step 0. Calculate $\delta_0(v, v', t)$ for all $t \in \mathcal{T}$ and $v, v' \in \mathcal{V}$ such that $(v, v') \in \mathcal{E}$ or $d_0(v', t)$ for all $t \in \mathcal{T}$ and $v' \in \mathcal{V}$, and calculate a lower bound $\hat{q}$. Set $i = 0, \mathcal{K}(i) = \{(v_0, 0, \infty)\}$.

Step 1. If $\mathcal{K}(i)$ is empty, replace i by $i - 1$. Else, go to Step 3.

Step 2. If $i = 0$, stop: the last saved path is optimal and $\hat{q}$ is its probability of detection. Else, go to Step 1.

Step 3. Remove from $\mathcal{K}(i)$ the triplet $(v_t, t, \bar{q}(v_t, t))$ with the largest bound $\bar{q}(v_t, t)$.

Step 4. If $\bar{q}(v_t, t) \leq \hat{q}$, go to Step 1. (Current subpath is fathomed.)

Step 5. If $i = 0$, replace i by $i + 1$, and go to Step 3. ($\mathcal{K}(1)$ is populated in the network reduction procedure.)

Step 6. If $t < T$, then for each vertex $v \in \mathcal{F}(v_t)$, calculate $\bar{q}(v, t+1)$ from (5) using bounds $d_0(v, t+1)$ or $\delta_0(v_t, v, t+1)$, and add $(v, t+1, \bar{q}(v, t+1))$ to $\mathcal{K}(i+1)$. Replace i by $i+1$, and go to Step 3. Else, let $\hat{q} = \bar{q}(v_t, t)$ and save the incumbent path $\{v_l\}_{l=0}^{T}$, and go to Step 1.

We now present the network reduction procedure that can be implemented as part of Step 0 of Algorithm 2. The procedure assumes that a static bound $d_0(v', t)$ is available as well as a lower bound on the optimal probability of detection $\hat{q}$. If the directional static bound is available instead of the static bound, replace $d_0(v', t)$ by $\delta_0(v, v', t)$ in the procedure below.

We note again that the network reduction procedure is valid under the assumption that $g(v_{s-1}, v_s, s)$ does not vary with the choice of $v_{s-1} \in \mathcal{R}(v_s)$ among all nodes-of-first-contact $\langle v_s, s\rangle$. We generalize this in Section 4.

Network Reduction Procedure R1.

Step 1. Find all nodes-of-first-contact $\langle v_s, s\rangle \in \mathcal{N}$. If none exist with $s > 1$, then stop.

Step 2. For each $\langle v_s, s\rangle$, calculate $\bar{q}(v_s, s) = p(\phi(v_s), s)g(v_{s-1}, v_s, s) + d_0(v_s, s)$.

Step 3. Eliminate all nodes-of-first-contact $\langle v_s, s\rangle$ with $\bar{q}(v_s, s) \leq \hat{q}$.

Step 4. For each nodes-of-first-contact $\langle v_s, s\rangle$ not eliminated, store the triplet $(v_s, s, \bar{q}(v_s, s))$ in $\mathcal{K}(1)$.

Table 2 illustrates the effect of the network reduction technique as applied to the same problem instances as in Table 1. Columns 5 and 6 of Table 2 present the run time and number of branching attempts, respectively, for Algorithm 2 with static bound and network reduction. On average, the run times and and the branching attempts are reduced to 5.3% and 5.0% of the corresponding numbers obtained without the network reduction technique (see columns 3 and 4 in Table 1), respectively. When applying both network reduction and directional static bound, we obtain the run times and numbers of branching attempts reported in columns 7 and 8 of Table 2. It is clear that network reduction and directional static bound have complementary positive effect and the run times and numbers of branching attempts are further reduced.

Table 3 presents computational results for a larger problem instance with 15 by 15 cells and a time horizon $T = 20$. Again, the searcher starts in the upper-left cell and the targets starts in the center cell. As seen from Table 3, the run times remain rather short for Algorithm 2 with directional static bound and network reduction (columns 3 and 4) while the times increases dramatically for Algorithm 1 with dynamic bound (columns 5 and 6). Furthermore, Algorithm 2 with directional static bound and network reduction is less sensitive to the detrimental effect of a near stationary target (e.g., $\rho = 0.9$) and high glimpse

detection probability (e.g., $g(v, v', t) = 0.9$ or 0.99), a case where the bounds tend to be weak.

The same problem instances were also examined in [17], which reports a run time of 10.9 seconds for the case with $g(v, v', t) = 0.6$ and $\rho = 0.6$ using a C++ implementation of Algorithm 1 with a dynamic bound running on a 2.6 GHz computer. We observe that our implementation appears to be somewhat slower than the one achieved in [17] with 30.47 seconds compared to 10.9 seconds (on a presumedly slightly slower computer). However, Algorithm 2 with directional static bound and network reduction appears to offer a noticeable advantage over Algorithm 1 with dynamic bound as derived in [17]. In principle, Algorithm 1 with dynamic bound can also be speeded up by using the proposed network reduction technique. However, we have not examined that possibility. We adopt a directional static bound with network reduction as the basis for extension to the case with side constraints discussed in the next section.

4 Algorithm for Resource-Constrained Search Problem

We now turn the attention to the full problem with side constraints, i.e., the resource-constrained search problem (RCSP) formulated in Section 2. We first develop a static bound based on Lagrangian relaxation that can be used within a branch-and-bound algorithm in the form of Algorithms 1 and 2. Second, we briefly discuss the development of a Lagrangian directional static bound. Third, we develop a series of network reduction techniques. Fourth, we combine the resulting procedures and present the complete algorithm.

4.1 Lagrangian Static Bound

Consider the time-expanded network $(\mathcal{N}, \mathcal{A})$, see Subsection 3.1, with the arc rewards $c_{n,n'}$ given in (15). Now, we also endow each arc $(n, n') \in \mathcal{A}$, $n = \langle v, t-1 \rangle$ and $n' = \langle v', t \rangle$, with weights $r_{i,n,n'} = f_i(v, v', t), i \in \mathcal{I}$. While computing the static bound in the case of no side constraints amounts to solving a longest-path problem on the time-expanded graph, a similar bound in the case with side constraints will need to account for those constraints.

Specifically, a static bound can be obtained by solving a constrained longest-path problem. We formulate this problem as an integer program on a slightly modified time-expanded graph.

We use the same time-expanded graph as in Subsection 3.1, with the following modifications. We recall that $\hat{\mathcal{V}}$ is the set of vertices where the search can end. Now, every node $n = \langle v, t \rangle$, $v \in \hat{\mathcal{V}}$, $t \in \mathcal{T}$ is connected to the artificial destination node $\hat{n}$ with an arc. This modification allows the searcher to terminate the search prior to time period T to avoid violating the side constraints. Furthermore, all arcs $(n, \hat{n})$ with $n = \langle v, T \rangle$, $v \notin \hat{\mathcal{V}}$, are removed from the time-expanded network. This modification makes the searcher return to $v \in \hat{\mathcal{V}}$. We still let $\mathcal{A}$ denote the set of all arcs.

We formulate the constrained longest-path problem on the time-expanded graph $(\mathcal{N}, \mathcal{A})$ as an integer program. We consider an ordering of $\mathcal{A}$ and let $\mathbf{A}$ denote the $|\mathcal{N}|$ by $|\mathcal{A}|$ node-arc incidence matrix for the time-expanded graph. For each arc $a = (n, n') \in \mathcal{A}$, let $A_{n,a} = 1$, $A_{n',a} = -1$, and $A_{n'',a} = 0$ for any $n'' \in \mathcal{N}, n'' \neq n, n'$ be the elements of $\mathbf{A}$. Let $\mathbf{b}$ denote the $|\mathcal{N}|$-vector with $b_{n_0} = 1$, $b_{\hat{n}} = -1$ and $b_n = 0$ for all $n \in \mathcal{N} \backslash \{n_0, \hat{n}\}$. We also define the additional notation: $\hat{\mathbf{r}} = (\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_I)^T$, where T denotes the transpose. We collect the rewards $c_{n,n'}$ in the $|\mathcal{A}|$-dimensional row vector $\mathbf{c}$. Moreover, for each $i \in \mathcal{I}$, we define the $|\mathcal{A}|$-dimensional row vector $\mathbf{r}_i$ to contain the weights $r_{i,n,n'}$ and we let $\mathbf{R}$ be the $|\mathcal{I}|$ by $|\mathcal{A}|$ matrix with $\mathbf{r}_i$ as its rows. Finally, we let $\mathbf{x}$ be a $|\mathcal{A}|$-dimensional column vector, where $x_{n,n'} = 1$ if arc (n, n') is used by a path, and zero otherwise. Then, the constrained longest-path problem, see [1], can be written as:

$$z^* \equiv \max_{\mathbf{x} \in \{0,1\}^{|\mathcal{A}|}} \mathbf{c}\mathbf{x} \tag{18}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{R}\mathbf{x} \leq \hat{\mathbf{r}}. \tag{19}$$

In principle, the solution of the constrained longest-path problem provides a static bound. However, the constrained longest-path problem is NP-complete even for the case with an acyclic graph ([13], pages 213-214). Hence, we prefer to avoid solving such problems within

an algorithm. We proceed by introducing an additional relaxation that is motivated by the solution approach for the constrained shortest-path problem in [14, 5, 6].

Using the standard theory of Lagrangian relaxation (see, e.g., [1], Chapter 16) and an $|\mathcal{I}|$-dimensional row vector $\boldsymbol{\lambda} \geq \mathbf{0}$, we find that

$$\begin{aligned} z^* \;\leq\; z(\boldsymbol{\lambda}) \;\equiv\; \max_{\mathbf{x}\in\{0,1\}^{|\mathcal{A}|}} \mathbf{cx} \;&-\; \boldsymbol{\lambda}(\mathbf{Rx} - \hat{\mathbf{r}}) \\ \text{s.t. } \mathbf{Ax} \;&=\; \mathbf{b}. \end{aligned} \tag{20}$$

Rewriting the objective function, we can optimize the Lagrangian upper bound $z(\boldsymbol{\lambda})$ through

$$\bar{z} \;\equiv\; \min_{\boldsymbol{\lambda}\geq\mathbf{0}} z(\boldsymbol{\lambda}) \tag{21}$$

$$\begin{aligned} = \;\min_{\boldsymbol{\lambda}\geq\mathbf{0}} \max_{\mathbf{x}\in\{0,1\}^{|\mathcal{A}}} &(\mathbf{c} - \boldsymbol{\lambda}\mathbf{R})\mathbf{x} + \boldsymbol{\lambda}\hat{\mathbf{r}} \\ \text{s.t. } \mathbf{Ax} \;&=\; \mathbf{b}. \end{aligned} \tag{22}$$

For any fixed $\boldsymbol{\lambda} \geq 0$, computing the upper bound $z(\boldsymbol{\lambda})$ simply requires the solution of a longest-path problem with Lagrangian-modified arc lengths in an acyclic graph. The outer minimization over $\boldsymbol{\lambda}$ can be solved by several methods [12, 3, 9, 5, 6]. Since we anticipate only a small number of side constraints, it suffices to use repeated coordinate search. Given an optimal or near-optimal $\boldsymbol{\lambda}$, we obtain the Lagrangian static bound by carrying out one backward longest-path calculation in the time-expanded graph from node $\hat{n}$ to all nodes $n \in \mathcal{N}$ using the Lagrangian modified arc reward $\mathbf{c} - \boldsymbol{\lambda}\mathbf{R}$. More specifically, an arc $(n, n') = (\langle v, s\rangle, \langle v', s+1\rangle) \in \mathcal{A}$, $s = 1, 2, ..., T-1$, is endowed with the reward

$$\tilde{c}_{n,n'} = c_{n,n'} - \sum_{i\in\mathcal{I}} \lambda_i r_{i,n,n'}, \tag{23}$$

where $c_{n,n'}$ is given by (15).

We also derive and implement a Lagrangian directional static bound similar to the one in Subsection 3.2.3. However, the derivation is a straightforward combination of Subsection 3.2.3 and the approach described above. Hence, we omit it. This derivation results in a similar Lagrangian problem to the one in (21), but now defined on the node-and-time expanded network. We still refer to the Lagrangian multiplier as $\boldsymbol{\lambda}$ and the Lagrangian

upper bound as $z(\boldsymbol{\lambda})$. We denote the Lagrangian directional static bound computed in this way by $\tilde{\delta}_0(v, v', t)$. Since the Lagrangian directional static bound is at least as strong as the Lagrangian static bound, we carry out the Lagrangian relaxation only in the node-and-time expanded network to find a Lagrangian multiplier $\boldsymbol{\lambda} \geq \mathbf{0}$.

4.2 Network Reduction

We propose and examine three techniques for reducing the size of the network prior to application of a branch-and-bound procedure. First, we use dominance rules to eliminate edges that cannot be on an optimal path. Second, we describe the application of "preprocessing" techniques frequently used prior to solving constrained shortest-path problems. Third, we modify the procedure described in Subsection 3.2.4.

4.2.1 Edge Dominance

There are several situation where a vertex $v' \in \mathcal{F}(v)$ can be eliminated as candidate for visit from vertex v. Such "dominance tests" are case dependent, but can be effective in reducing the number of edges. We describe one situation where we use "edge dominance."

In many practical situation, there are two resources: risk and fuel. If higher altitude implies lower risk and lower glimpse detection probability, and climbing to higher altitude consumes more fuel than level flight, then we can eliminate some edges in the graph $(\mathcal{V}, \mathcal{E})$ by "edge dominance." Suppose that $f_1(v, v', t)$ and $f_2(v, v', t)$ represent risk and fuel, respectively. Also suppose that the risk $f_1(v, v', t) = f_1(v')$, i.e., only depends on v'. Let $\psi(v)$ be the altitude of waypoint $v \in \mathcal{V}$. Then, if we have the above described situation, we use the following (one-step) procedure to reduce the size of the graph $(\mathcal{V}, \mathcal{E})$.

Edge Dominance Procedure R2.

Step 1. Delete any edge $(v, v') \in \mathcal{E}$ that satisfies $f_1(v') = 0$ and $\psi(v) < \psi(v')$.

We note that Procedure R2 takes advantage of the fact that if there is no risk at v', then there is no need to increase altitude when moving from v to v'. The altitude can be increased later if need be.

4.2.2 Preprocessing

It is well known that the side constraints (19) can be used, prior to any main calculations, to identify nodes and arcs in the time-expanded network $(\mathcal{N}, \mathcal{A})$ that cannot lie on any feasible path [2, 10, 5, 6]. Such nodes and arcs can be eliminated from $(\mathcal{N}, \mathcal{A})$, which reduces the size of the problem that needs to be considered when computing bounds and carrying out branching. Moreover, the reduction of the time-expanded network typically also strengthens the Lagrangian relaxation, i.e., reduces the gap $z^* - \bar{z}$, see (21) and (20), and, hence, reduces the need for branching. We adopt the follow procedure, adapted from [5, 6], to carry out arc preprocessing:

Preprocessing Procedure R3.

Step 1. Set number of iterations $\bar{k}$ and $k = 1$.

Step 2. For all $i \in \mathcal{I}$ and $n \in \mathcal{N}$, compute a minimum-weight n_0-n subpath distance $\underline{R}_i(n)$ and a minimum-weight n-$\hat{n}$ subpath distance $\underline{r}_i(n)$ in $(\mathcal{N}, \mathcal{A})$ with respect to weights $r_{i,n,n'}$.

Step 3. Delete any arc $(n, n') \in \mathcal{A}$ with

$$\underline{R}_i(n) + r_{i,n,n'} + \underline{r}_i(n') > \hat{r}_i \text{ for any } i \in \mathcal{I}. \tag{24}$$

Step 4. If $k < \bar{k}$ and at least one arc was deleted in Step 3, replace k by $k + 1$, and go to Step 2. Else, stop.

If a lower bound on the probability of detection is available, we also carry out similar preprocessing with respect to arc reward $c_{n,n'}$ and the Lagrangian modified arc reward $\tilde{c}_{n,n'} = c_{n,n'} - \sum_{i \in \mathcal{I}} \lambda_i r_{i,n,n'}$, see [5, 6].

We describe the preprocessing procedure for the time-expanded network and argue that it improves the Lagrangian static bound. However, the same methodology applies to the node-and-time expanded network and it improves the Lagrangian directional static bound. In our main algorithm (described in Subsection 4.3), we also apply preprocessing to the node-and-time expanded network and denote that procedure R3′.

4.2.3 Vertex Dominance for Distant Target

As in Subsection 3.2.4, we consider the case where the searcher's and the target's locations are initially some distance apart and derive a network reduction procedure that utilizes that situation. In contrast to Subsection 3.2.4, the subpath used to reach a node-of-first-contact is now important since the resource consumption along different subpaths may be different. Hence, $\langle v_0, 0\rangle$ now needs to be connected to each node-of-first-contact $\langle v_s, s\rangle$ with multiple "jump" arcs representing the different possible subpaths and resource consumptions used to reach $\langle v_s, s\rangle$. A standard path enumeration algorithm (see, e.g., [7]) can enumerate the different subpaths, at least as long as s is relatively small. Multiple arcs to a node-of-first-contact can also be used to model the situation with edge-dependent glimpse detection probability, a case ignored in Subsection 3.2.4.

After all the arcs are generated to all the nodes-of-first-contact, a number of them can be deemed uninteresting and be eliminated using dominance rules of the form: If an arc from $\langle v_0, 0\rangle$ to $\langle v_s, s\rangle$ has no larger probability of detection and no smaller consumption of each resource as another parallel arc and the two arcs are not identical, the first arc is dominated and can be eliminated. In sets of identical parallel arcs, we also eliminate all but one. Trivially, arcs with resource consumption greater than the specified limits are also removed. Moreover, if a lower bound on the optimal probability of detection exists, it can be used to eliminate more arcs as described in the following.

Below we describe this network reduction procedure based on vertex dominance for distant target formally. We note that the procedure is more effective after (i) applying network reduction procedures R2, R3 and R3′, (ii) finding $\boldsymbol{\lambda}$ that (approximately) optimizes the Lagrangian upper bound $z(\boldsymbol{\lambda})$ on the node-and-time expanded network, and (iii) computing the directional static bound $\delta_0(v, v', t)$ and the Lagrangian directional static bound $\tilde{\delta}_0(v, v', t)$ for each node $\langle v', t\rangle \in \mathcal{N}$. So we assume that these calculations have been carried out. During these calculations, feasible paths may be obtained. Such paths provide lower bounds on the optimal probability of detection. Let $\hat{q}$ denote the largest lower bound found so far.

Vertex Dominance Procedure R4.

Step 1. Find all nodes-of-first-contact $\langle v_s, s\rangle \in \mathcal{N}$. If none exist with $s > 1$, then stop.

Step 2. For each node-of-first-contact $\langle v_s, s\rangle$, enumerate all subpaths $\langle v_0, 0\rangle$ to $\langle v_s, s\rangle$.

Step 3. For each node-of-first-contact $\langle v_s, s\rangle$ and subpath $\mathcal{P} = \{v_l\}_{l=0}^{s}$, carry out the tests:

If any of the following is true, then eliminate $\mathcal{P}$:

(i) For some remaining $\langle v_0, 0\rangle$-$\langle v_s, s\rangle$ subpath $\mathcal{P}'$, $r_i(\mathcal{P}) \geq r_i(\mathcal{P}')$ for all $i \in \mathcal{I}$ and $q(\mathcal{P}) \leq q(\mathcal{P}')$.

(ii) $q(\mathcal{P}) + \delta_0(v_{s-1}, v_s, s) \leq \hat{q}$.

(iii) $r_i(\mathcal{P}) + \underline{r}_i(\langle v_s, s\rangle) > \hat{r}_i$ for some $i \in \mathcal{I}$.

Step 3 can also be augmented with a test on the Lagrangian-modified probability of detection using $\tilde{\delta}_0(v, v', t)$ if a near-optimal multiplier $\boldsymbol{\lambda}$ is available.

4.3 Algorithm

We now state the complete algorithm for RCSP. The algorithm starts with network reductions procedures R2, R3, and R3′. The next step solves the Lagrangian problem (22) and determines a near-optimal $\boldsymbol{\lambda}$. (The calculations are actually carried out in the node-and-time expanded network as we prefer to use the Lagrangian directional static bound.) If a feasible path becomes available during the procedures described above, network reduction procedure R3′ is repeated now using checks with respect to arc reward $c_{\xi,\xi'}$ and its Lagrangian modified arc reward. The next steps are to compute the directional static bound of Subsection 3.2.3 (i.e., $\delta_0(v, v', t)$), the Lagrangian directional static bound as described in Subsection 4.1 (i.e., $\tilde{\delta}_0(v, v', t)$), and bounds on resource consumption along any path extension (i.e., $\underline{r}_i(n)$, $i \in \mathcal{I}$). The final steps before the branch-and-bound procedure is to implement network reduction procedure R4.

We implement the branch-and-bound procedure as an implicit path-enumeration in the time-expanded network. The procedure amounts to a depth-first search coupled with opti-

mality and feasibility checks using the computed bounds. The complete algorithm takes the following form:

Algorithm 3.

Step 1. Apply network reduction procedures R2, R3 and R3′.

Step 2. Find $\boldsymbol{\lambda}$ that approximately optimizes the Lagrangian upper bound $z(\boldsymbol{\lambda})$ in the node-and-time expanded graph (Ξ, Ω). If a feasible solution is found, set the probability of detection on the corresponding path equal to $\hat{q}$. Otherwise, set $\hat{q} = -\infty$.

Step 3. If a feasible solution is found so far, implement R3′ also with respect to arc reward and Lagrangian modified arc reward using $\hat{q}$.

Step 4. Ignoring side constraints, compute the directional static bound $\delta_0(v, v', t)$ for all expanded nodes $\xi = \langle n, n' \rangle$ in (Ξ, Ω), with $n = \langle v, t-1 \rangle$, $n' = \langle v', t \rangle$, $v, v' \in \mathcal{V}$.

Step 5. Using $\boldsymbol{\lambda}$ from Step 2, compute the Lagrangian directional static bound $\tilde{\delta}_0(v, v', t)$ for all expanded nodes $\xi = \langle n, n' \rangle$ in (Ξ, Ω), with $n = \langle v, t-1 \rangle$, $n' = \langle v', t \rangle$, $v, v' \in \mathcal{V}$.

Step 6. For each $i \in \mathcal{I}$, compute the minimum distance $\underline{r}_i(n)$ from each node $n \in \mathcal{N}$ back to $\hat{n}$ by solving a single, backwards, shortest-path problem in the time-expanded graph $(\mathcal{N}, \mathcal{A})$ starting from $\hat{n}$ using arc length $r_{i,n,n'}$.

Step 7. Apply network reduction procedure R4.

Step 8. Apply a standard path-enumeration procedure (see, e.g., [7]) in $(\mathcal{N}, \mathcal{A})$ with the following modifications:

(i) The path-enumeration commences from n_0, but extends a current subpath $\{n_l\}_{l=0}^{t}$ along arc $(n_t, n) = (\langle v_t, t \rangle, \langle v, t+1 \rangle)$ if and only if the following conditions hold:

- For all $i \in I$, $\{n_0, n_1, ..., n_t, n\}$ can be extended to a path whose i-th resource does not exceed $\hat{r}_i$, i.e.,

$$r_i(\{n_0, n_1, ..., n_t, n\}) + \underline{r}_i(n) \leq \hat{r}_i. \tag{25}$$

- $\{n_0, n_1, ..., n_t, n\}$ can be extended to a path with probability of detection exceeding $\hat{q}$, i.e.,

$$q(\{n_0, n_1, ..., n_t, n\}) + \delta_0(v_t, v, t+1) > \hat{q}. \tag{26}$$

- $\{n_0, n_1, ..., n_t, n\}$ can be extended to a path whose Lagrangian-modified probability is no less than $\hat{q}$, i.e.,

$$q(\{n_0, n_1, ..., n_t, n\}) - \sum_{i \in \mathcal{I}} \sum_{l=1}^{t} \lambda_i r_{i,n_{l-1},n_l} - \sum_{i \in \mathcal{I}} \lambda_i r_{i,n_t,n} + \boldsymbol{\lambda}\hat{\mathbf{r}} + \tilde{\delta}_0(v_t, v, t+1) \geq \hat{q}. \tag{27}$$

(ii) Whenever the algorithm identifies a path $\mathcal{P}$ with $q(\mathcal{P}) > \hat{q}$ and $r_i(\mathcal{P}) \leq \hat{r}_i, i \in \mathcal{I}$, replace $\hat{q}$ by $q(\mathcal{P})$.

In Step 8, the checks (25), (26), and (27) prevent the enumeration of paths that can be determined, using the computed bounds, to not be optimal. Specifically, (25) prevents the extension of subpaths that cannot result in a feasible path with respect to the side constraints. Since $\delta_0(v_t, v, t+1)$ is a valid upper bound on the probability of detection during time period $t+2$, $t+3$, ..., T, the left-hand side of (26) is an upper bound on the probability of detection along any path that starts with the subpath $\{n_0, n_1, ..., n_t, n\}$. Hence, the subpath cannot be extended to a path with larger probability of detection than $\hat{q}$ if (26) fails. In (27), the probability of detection along $\{n_0, n_1, ..., n_t, n\}$ is modified by Lagrangian terms and the Lagrangian directional static bound is used. The resulting check can be shown to be valid using standard Lagrangian relaxation theory and the argument above.

We also implement Step 8 with a "branching strategy" based on the Lagrangian directional static bound. Specifically, we first consider extending the current subpath $\{n_l\}_{l=0}^{t}$ along the arc (n_t, n) with the largest Lagrangian directional static bound among all the nodes in the forward star of n_t. Second, we consider extending $\{n_l\}_{l=0}^{t}$ with the node corresponding to the second largest Lagrangian directional static bound, etc. This branching strategy is analogous to the one in Step 3 of Algorithms 1 and 2. We also experimented with using the

directional static bound instead of the Lagrangian directional static bound and found it to usually result in comparable run times. However, the Lagrangian directional static bound appears faster, on average.

Since Algorithm 3, in the worst case, enumerates all feasible paths, it is guaranteed to find an optimal solution of RCSP.

5 Numerical Example

This section describes computational experiments with Algorithm 3 applied to RCSP with side constraints on risk exposure and fuel consumption. We carry out all experiments on the same computational platform as in Section 3.

We consider a military planning situation where a UAV is assigned a mission to search and detect a high-value moving target. Planners wish to determine a flight path over the area of interest (AOI) that maximizes the probability of detecting the target. The UAV will start its path at a known waypoint with a known fuel tank, and will return to the same waypoint before the fuel tank is empty. Doctrine specifies that the UAV cannot be assigned a path with higher risk than a specific threshold. The AOI is partially under enemy control and any aircraft flying over the AOI could be shot down by enemy surface-to-air missiles (SAMs), anti-aircraft artillery, and small-arms fire. Flying at a high altitude would reduces that risk, but it will also reduce the quality of the UAV's sensor. Consequently, the UAV may change altitude during the course of the mission to balance risk and sensor quality. Changing from low to high altitude consumes more fuel than level flight. Hence, the number of time periods available for search depends on the fuel consumption and therefore the vertical flight profile.

We model this situation by dividing the AOI into 10 by 10 cells (see Figure 6). The airspace over each cell is vertically discretized into two altitudes ("low" and "high"). The heavily shaded cells ($\mathcal{C}_1$) in Figure 6 represent an urban area over which the UAV's risk is high and its glimpse detection probability is low. The unshaded cells ($\mathcal{C}_3$) represent open terrain where there is no risk and the UAV's glimpse detection probability is high. The lightly shaded cells ($\mathcal{C}_2$) represent an area with intermediate risk and glimpse detection probability.

We assume that the risks at different edges along a path are independent. If the probability of the UAV surviving edge $(v, v') \in \mathcal{A}$ is $\sigma(v, v')$, then the probability of surviving the path $\{v_t\}_{t=0}^{k}$ is simply $\Pi_{l=1}^{k}\sigma(v_{l-1}, v_l)$. Let $\hat{\sigma}$ be a lower limit on the survival probability. Then, a standard logarithmic transformation leads to the following constraint

$$\sum_{l=1}^{k} -\log \sigma(v_{l-1}, v_l) \leq -\log \hat{\sigma}, \tag{28}$$

which is in the form (3) and (4) with $f_i(v_{l-1}, v_l, l) = -\log \sigma(v_{l-1}, v_l)$ and $\hat{r}_i = -\log \hat{\sigma}$.

For this computational experiment, we assume that the glimpse detection probability and the survival probability for an edge $(v, v') \in \mathcal{E}$ depend only on the cell and altitude corresponding to vertex $v' \in \mathcal{V}$ as listed in Table 4. We note that glimpse detection probability at high altitude is assumed to be 70% of the one at low altitude and the failure probability (complement of the survival probability) at high altitude is 30% of the one at low altitude.

The UAV enters the airspace at high altitude over the northwest cell (cell 1; cells are numbered from left to right, and from top to bottom) and will return to the same cell at either high or low altitude at the end of the mission. The searcher is located at one vertex each time period and searches the corresponding cell. For the next time period, the searcher can stay at the same vertex, change altitude over the same cell, or move to a vertex (at any altitude) corresponding to a vertically or horizontally adjacent cell. The maximum number of time periods is $T = 40$, but the fuel consumption constraint may limit the number of periods to less than 40. We assume that the fuel consumption at each time step is as follows: 10 units if there is no altitude change, 12(9) units if changing from low(high) altitude to high(low) altitude. The initial position of the target is the center of the high risk region (cell 68). The target remains in the current cell with a probability $\rho = 0.6$ for the next time period or moves to one of the vertically or horizontally adjacent cells with equal probability.

The survival probability limit is a threshold that is set by the commander or planner. A search path with lower survival probability than the threshold would not be accepted. In this experiment, we consider the survival probability limits 0.95, 0.90,...,0.75, and 0.70, and fuel consumption limit 300, 325, ..., 425, and 450.

We solve this problem instance using Algorithm 3. Tables 5, 6, and 7 report computational results for different combinations of survival probability and fuel limits for the UAV. When the fuel limit is tight (e.g., 300 and 325), the UAV cannot operate for the full duration of 40 times steps. We observe that increasing the fuel limit beyond 425 do not increase the probability of detection as the time limit of 40 periods becomes active. The average run time is 580 seconds, with a standard deviation of 792. All problem instances are solved within one hour and typically in much less. Figure 7 shows the optimal path given survival probability limit 0.90 and fuel limit 400. The solid lines and the dashed lines represent flight segment at low and high altitude, respectively.

We also consider the case with edge-dependent glimpse detection probability. Consider the same situation as earlier described, but now assume that a move to a new waypoint results in a lower glimpse detection probability than if the searcher already was at that waypoint. Specifically, if $v = v'$, we let the glimpse detection probability $g(v, v', t)$ be as in Table 4; otherwise we replace $g(v, v', t)$ by $0.1g(v, v', t)$. Figure 8 shows an optimal path found given survival probability and fuel limits of 0.90 and 400, respectively. In contrast to the case with edge-independent glimpse detection probability (Figure 7), the searcher now tends to stay for multiple time periods at the same waypoints in high-probability regions to reap the benefits of the corresponding high glimpse detection probability. The run times (not reported in detail) for the case with edge-dependent glimpse detection probabilities are, on average, 53 seconds, with a standard deviation of 71 seconds. The reduction in run time compared to the edge-independent case is caused by the often lower glimpse detection probability ($0.1g(v, v', t)$), which tightens the bound.

6 Conclusions

This paper formulates the resource-constrained search problem, which generalized existing search models by considering (i) history-dependent glimpse detection probability, (ii) multiple altitudes for the searcher, and (iii) multiple constraints on "consumption" of resources such as time, fuel, and risk. We develop a specialized branch-and-bound algorithm for the

solution of the resource-constrained search problem. We propose a new bound on the optimal probability of detection using network expansion to account for a portion of the history of the current path and Lagrangian relaxation to eliminate resource constraints. After the Lagrangian multiplier vector is optimized, the bound is computed using a single, backward longest-path calculation in an acyclic graph. We also derive a series of network reduction procedures that tighten the Lagrangian relaxation and reduces the amount of enumeration.

In direct comparison with a state-of-the-art algorithm for the time-constrained search problem, the proposed bound and network reduction procedures reduce the run times with at least an order of magnitude. In more complicated resource-constrained search problem with time, fuel, and risk constraints as well as two altitudes, our algorithm solves realistic instances typically within about 20 minutes.

Acknowledgement

The authors thank Professors Kevin Wood, Moshe Kress, and Alan Washburn, Naval Postgraduate School, for valuable comments and Dr. Haye Lau, University of Technology in Sydney, for assistance with implementation of Algorithm 1 with dynamic bound as closely as possible to his original implementation in [17].

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, Upper Saddle River, New Jersey, 1993.

[2] Y. Aneja, V. Aggarwal, and K. Nair. Shortest chain subject to side conditions. *Networks*, 13:295–302, 1983.

[3] J. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[4] T. Caldwell. On finding minimal routes in a network with turning penalties. *Communications of the ACM*, 4:107–108, 1961.

[5] W. M. Carlyle, J. O. Royset, and R. K. Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks*, to appear: Available at www.nps.navy.mil/orfacpag/resumePages/papers/roysetpa.htm, 2007.

[6] W. M. Carlyle, J. O. Royset, and R. K. Wood. Routing military aircraft with a constrained shortest-path algorithm. *Military Operations Research*, in review: Available at www.nps.navy.mil/orfacpag/resumePages/papers/roysetpa.htm, 2007.

[7] W. M. Carlyle and R. K. Wood. Near-shortest and k-shortest simple paths. *Networks*, 46:98–109, 2005.

[8] R.F. Dell, J.N. Eagle, G.H.A. Martins, and A.G. Santos. Using multiple searchers in constrained-path, moving-target search problems. *Naval Research Logistics*, 43:463–480, 1996.

[9] D. DeWolfe, J. Stevens, and R. K. Wood. Setting military reenlistment bonuses. *Naval Research Logistics*, 40:143–160, 1993.

[10] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithm for the weight-constrained shortest path problem. *Networks*, 42:135–153, 2003.

[11] J.N. Eagle and J.R. Yee. An optimal branch and bound procedure for the constrained path, moving target search problem. *Operations Research*, 38:110–114, 1990.

[12] B. L. Fox and D. M. Landi. Searching for the multiplier in one-constraint optimization problems. *Operations Research*, 18:253–262, 1970.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, California, 1979.

[14] G. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[15] N. J. Karczcewski. *Optimal aircraft routing in a constrained path-dependent environment*. Master's thesis, Naval Postgraduate School, Monterey, California, 2007.

[16] M. Kress and J. O. Royset. Aerial search optimization model (ASOM) for uavs in special operations. *Military Operations Research, to appear*, 2007.

[17] H. Lau, S. Huang, and G. Dissanayake. Discounted mean bound for the optimal searcher path problem with non-uniform travel times. *European Journal of Operational Research*, in press: Published online, 2007.

[18] G. A. Martin. *A New Branch-and-bound Procedure for Computing Optimal Search Path.* Master's thesis, Naval Postgraduate School, Monterey, California, 1993.

[19] R. Murphey, S. Uryasev, and M. Zabarankin. Optimal path planning in a threat environment. In P. Pardalos, editor, *Recent Developments in Cooperative Control and Optimization*, pages 349–406, Kluwer Academic, Dordrecht, Netherlands, 2003.

[20] D.N. Reber. Major, Ops Analysis Division, U.S. Marine Corps Combat Development Command. Private communication November 26, 2007.

[21] T. J. Stewart. Search for a moving target when searcher motion is restricted. *Computers & operations research*, 6(3):129–140, 1979.

[22] K. E. Trummel and J.R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.

[23] A. R. Washburn. Search for a moving target: The fab algorithm. *Operations Research*, 31(4):739–751, 1983.

[24] A. R. Washburn. Branch and bound methods for search problems. Technical report, Naval Postgraduate School, Monterey, California, April 1995.

[25] A. R. Washburn. Branch and bound methods for a search problem. *Naval Research Logistics*, 45:243–257, 1998.

[26] M. Zabarankin, S. Uryasev, and R. Murphey. Aircraft routing under the risk of detection. *Naval Research Logistics*, 53:728–747, 2006.

		Static Bound		Dynamic Bound [17]			Dynamic Bound	
		Time	Branching	Time	Scaled	Branching	Time	Branching
$g(v, v', t)$	ρ	(sec.)	attempts	(sec.)	(sec.)	attempts	(sec.)	attempts
	0.3	3.59	2,301,182	14.56	0.71	58,314	2.09	58,314
0.3	0.6	2.58	1,635,517	14.53	0.71	52,369	2.11	52,369
	0.9	11.47	7,338,492	157.30	7.71	380,889	20.75	380,889
	0.3	5.38	3,424,282	19.07	0.94	49,774	2.59	49,774
0.6	0.6	2.58	1,620,402	23.76	1.16	47,454	3.03	47,454
	0.9	59.26	38,186,809	730.96	35.84	2,185,066	103.08	2,185,066
	0.3	5.78	3,675,197	21.55	1.06	45,019	2.78	45,019
0.9	0.6	2.89	1,831,875	30.58	1.50	59,527	3.91	59,527
	0.9	176.26	113,646,357	2902.27	142.30	11,299,259	431.38	11,299,259
	0.3	6.09	3,865,002	N/A	N/A	N/A	2.91	46,339
0.99	0.6	3.00	1,896,960	N/A	N/A	N/A	3.91	58,752
	0.9	192.06	123,822,672	N/A	N/A	N/A	558.63	16,685,969

Table 1: Run times and number of branching attempts for Algorithm 1 with static and dynamic bounds on 11 by 11 cell search problem with time horizon $T = 17$. Columns labeled "Dynamic Bound [17]" correspond to original and speed-adjusted results from [17].

$g(v, v', t)$	ρ	Algo. 1: D-Static Time (sec.)	Branching attempts	Algo. 2: Static & Red. Time (sec.)	Branching attempts	Algo. 2: D-Static & Red. Time (sec.)	Branching attempts
	0.3	2.59	1,642,619	0.22	129,990	0.19	91,198
0.3	0.6	1.80	1,125,929	0.17	94,307	0.16	65,485
	0.9	6.30	4,032,951	0.58	354,677	0.41	223,435
	0.3	3.50	2,245,784	0.31	194,425	0.27	128,526
0.6	0.6	1.45	902,409	0.17	92,997	0.14	57,015
	0.9	29.67	19,321,387	2.17	1,442,612	1.37	844,747
	0.3	3.59	2,282,989	0.34	209,160	0.28	138,598
0.9	0.6	1.66	1,037,829	0.17	101,216	0.17	61,005
	0.9	80.50	52,527,302	4.95	3,323,101	2.89	1,837,647
	0.3	3.77	2,396,764	0.36	219,217	0.28	137,063
0.99	0.6	1.72	1,080,459	0.19	102,763	0.17	62,890
	0.9	87.99	57,427,410	5.39	3,592,696	3.09	1,974,871

Table 2: Run time and number of branching attempt for Algorithm 1 on problem instances of Table 1 using directional static bound (D-Static) and Algorithm 2 using static bound and network reduction (Static & Red.) and directional static bound and network reduction (D-Static & Red.).

$g(v, v', t)$	ρ	Algo. 2: D-Static & Red. Time (sec.)	Branching attempts	Algo. 1: Dynamic Bound Time (sec.)	Branching attempts
	0.3	3.08	103,811	20.24	328,672
0.3	0.6	2.94	66,185	21.22	311,645
	0.9	3.58	238,089	107.11	1,352,503
	0.3	3.14	122,941	20.28	248,727
0.6	0.6	2.92	51,977	30.47	288,738
	0.9	4.41	571,649	701.17	8,668,034
	0.3	3.17	129,276	29.61	292,818
0.9	0.6	2.95	57,353	45.05	404,299
	0.9	5.94	1,076,948	2323.48	30,173,994
	0.3	3.13	128,776	31.97	301,498
0.99	0.6	2.92	60,449	48.42	441,664
	0.9	5.77	1,016,918	3092.63	45,329,829

Table 3: Run times and number of branching attempts for Algorithm 2 with directional static bound and network reduction compared with Algorithm 1 with dynamic bound on 15 by 15 cell search problem with time horizon $T = 20$.

Cell	Altitude	Glimpse probability	Survival probability
$\mathcal{C}_1$	low	0.20	0.960
	high	0.14	0.988
$\mathcal{C}_2$	low	0.40	0.980
	high	0.28	0.994
$\mathcal{C}_3$	low	0.60	1.000
	high	0.42	1.000

Table 4: Glimpse detection probability $g(v, v', t)$ and survival probability $\sigma(v, v')$ for different cells and altitude.

	Survival prob. limit = 0.95					Survival prob. limit = 0.90			
Fuel limit	Prob. Detection	Survival Prob.	Fuel	Run time (sec.)	Fuel limit	Prob. Detection	Survival Prob.	Fuel	Run time (sec.)
300	0.131501	0.962466	300	26.89	300	0.135098	0.915157	300	29.58
325	0.153228	0.952892	321	98.20	325	0.166933	0.901925	322	32.66
350	0.176511	0.952892	350	3313.64	350	0.194440	0.915157	350	115.94
375	0.204686	0.952892	371	661.43	375	0.217277	0.901925	372	84.00
400	0.236651	0.962466	400	664.96	400	0.246767	0.902268	400	537.71
425	0.237081	0.952892	401	2721.68	425	0.249996	0.901925	402	361.60
450	0.237081	0.952892	401	2724.22	450	0.249996	0.901925	402	361.52

Table 5: Computational results for Algorithm 3. Survival probability limit = 0.95 and 0.90.

	Survival prob. limit = 0.85					Survival prob. limit = 0.80			
Fuel limit	Prob. Detection	Survival Prob.	Fuel	Run time (sec.)	Fuel limit	Prob. Detection	Survival Prob.	Fuel	Run time (sec.)
300	0.145309	0.851528	300	25.31	300	0.145809	0.805953	299	22.03
325	0.173000	0.851528	320	37.81	325	0.173218	0.839535	319	37.39
350	0.203183	0.851528	350	67.69	350	0.204891	0.805953	349	64.17
375	0.222393	0.851528	370	115.24	375	0.223377	0.805953	369	116.69
400	0.255481	0.851528	400	510.68	400	0.255813	0.827710	399	880.65
425	0.255481	0.851528	400	508.50	425	0.255813	0.827710	399	880.40
450	0.255481	0.851528	400	508.96	450	0.255813	0.827710	399	880.46

Table 6: Computational results for Algorithm 3. Survival probability limit = 0.85 and 0.80.

	Survival prob. limit = 0.75					Survival prob. limit = 0.70			
Fuel limit	Prob. Detection	Survival Prob.	Fuel	Run time (sec.)	Fuel limit	Prob. Detection	Survival Prob.	Fuel	Run time (sec.)
300	0.150092	0.753377	300	21.33	300	0.150277	0.742767	299	21.38
325	0.175850	0.753377	320	37.50	325	0.176068	0.742767	319	31.16
350	0.205935	0.753377	350	63.84	350	0.206200	0.742767	349	53.42
375	0.223703	0.753377	370	125.97	375	0.224003	0.713056	369	121.33
400	0.255813	0.827710	399	1220.30	400	0.255813	0.827710	399	1280.52
425	0.255813	0.827710	399	1217.21	425	0.255813	0.827710	399	1277.52
450	0.255813	0.827710	399	1218.91	450	0.255813	0.827710	399	1282.42

Table 7: Computational results for Algorithm 3. Survival probability limit = 0.75 and 0.70.

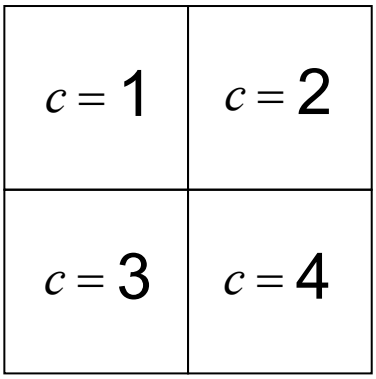

Figure 1: A discretized area of interest composing of $C = 4$ cells.

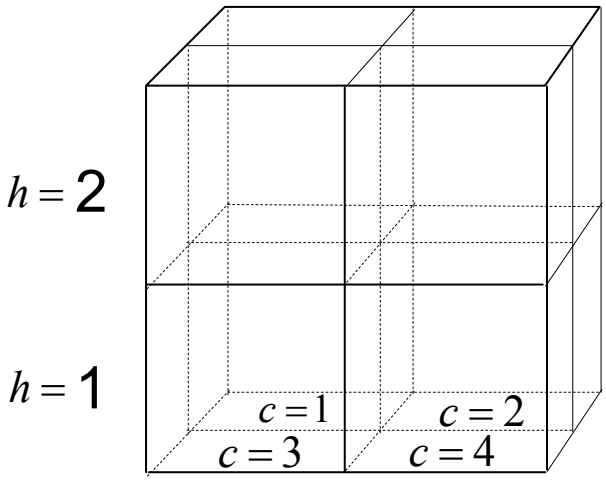

Figure 2: A discretized airspace over the area of interest (Figure 1) with $H = 2$ altitudes.

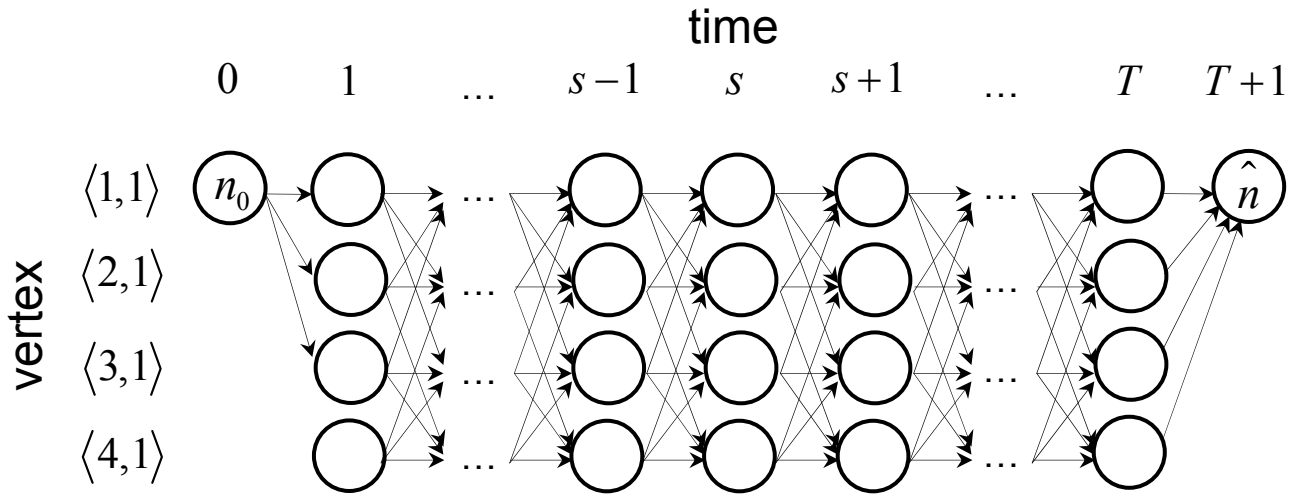

Figure 3: A time-expanded graph from the network in Figure 1 and one altitude. The searcher's prior position is $n_0 = \langle v_0, 0\rangle = \langle\langle 1, 1\rangle, 0\rangle$ and final position is $\hat{n} = \langle \hat{v}, T + 1\rangle$.

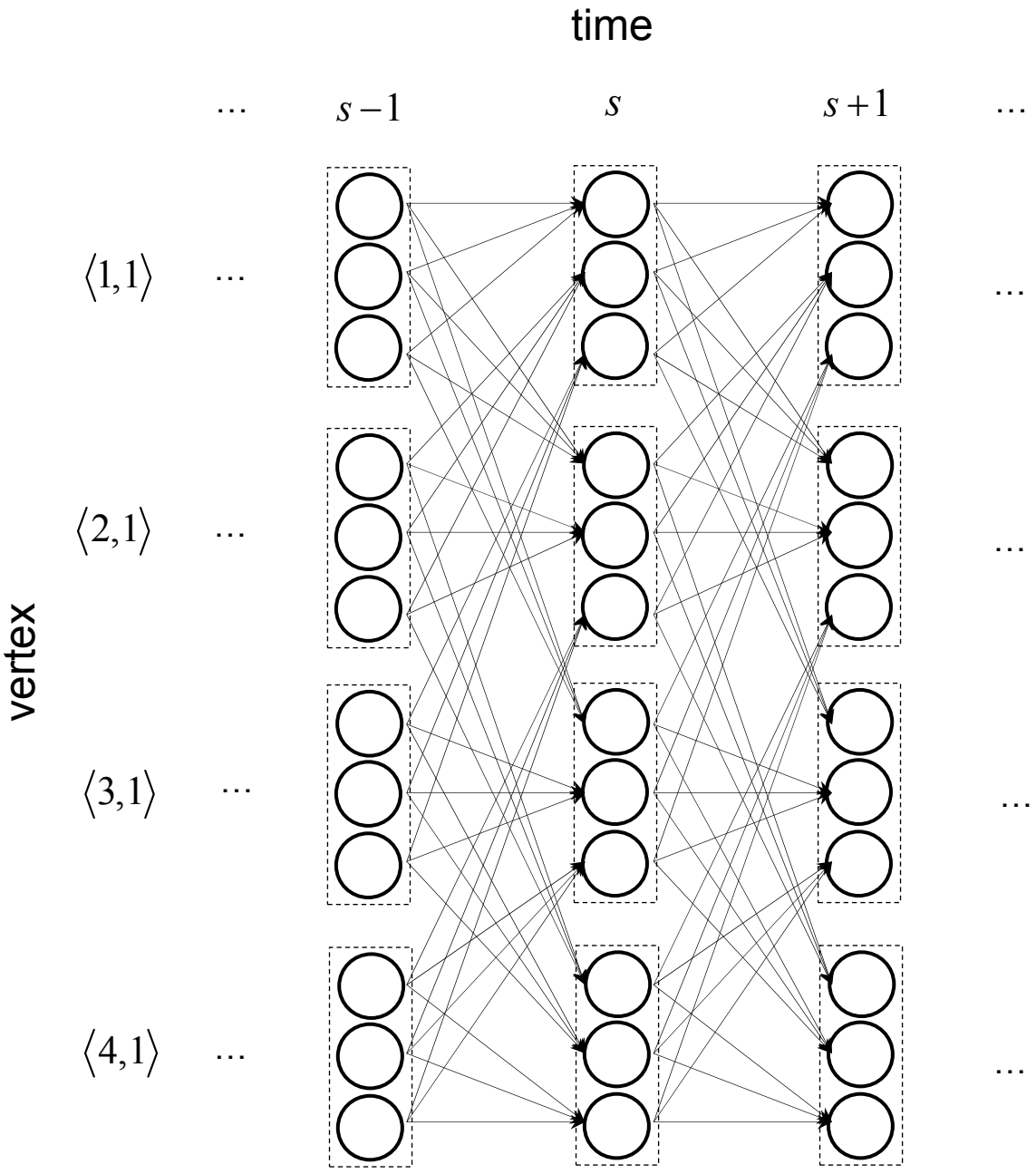

Figure 4: A node-and-time expanded network from the time-expanded graph (Figure 3).

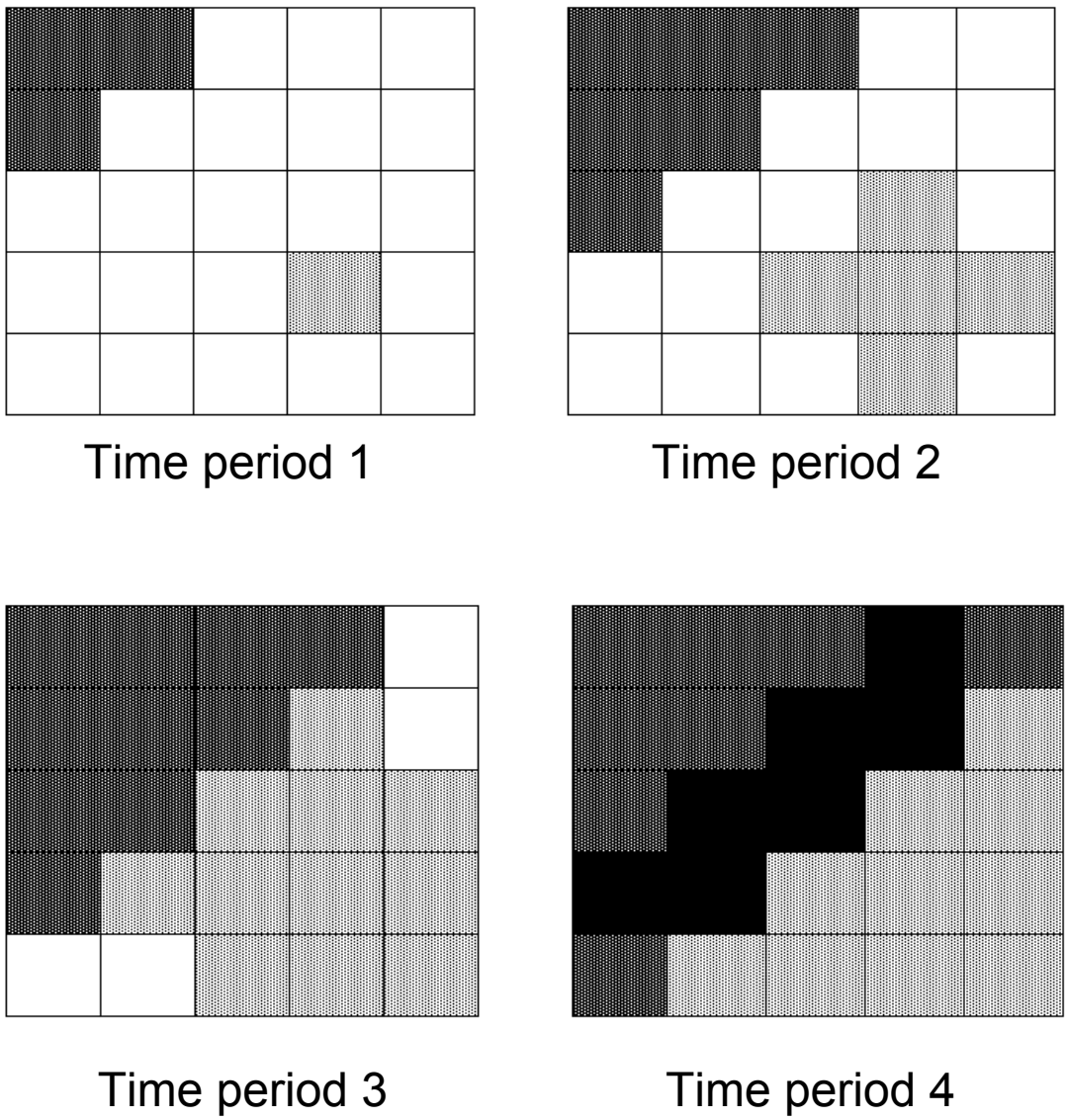

Figure 5: An area of interest composing of 5 by 5 cells. For each time period, the unshaded, lightly-shaded, heavily-shaded, and completely-shaded cells describe the regions where neither searcher nor target stay, only target possibly stays, only searcher possibly stays, and target and searcher possibly stay, respectively.

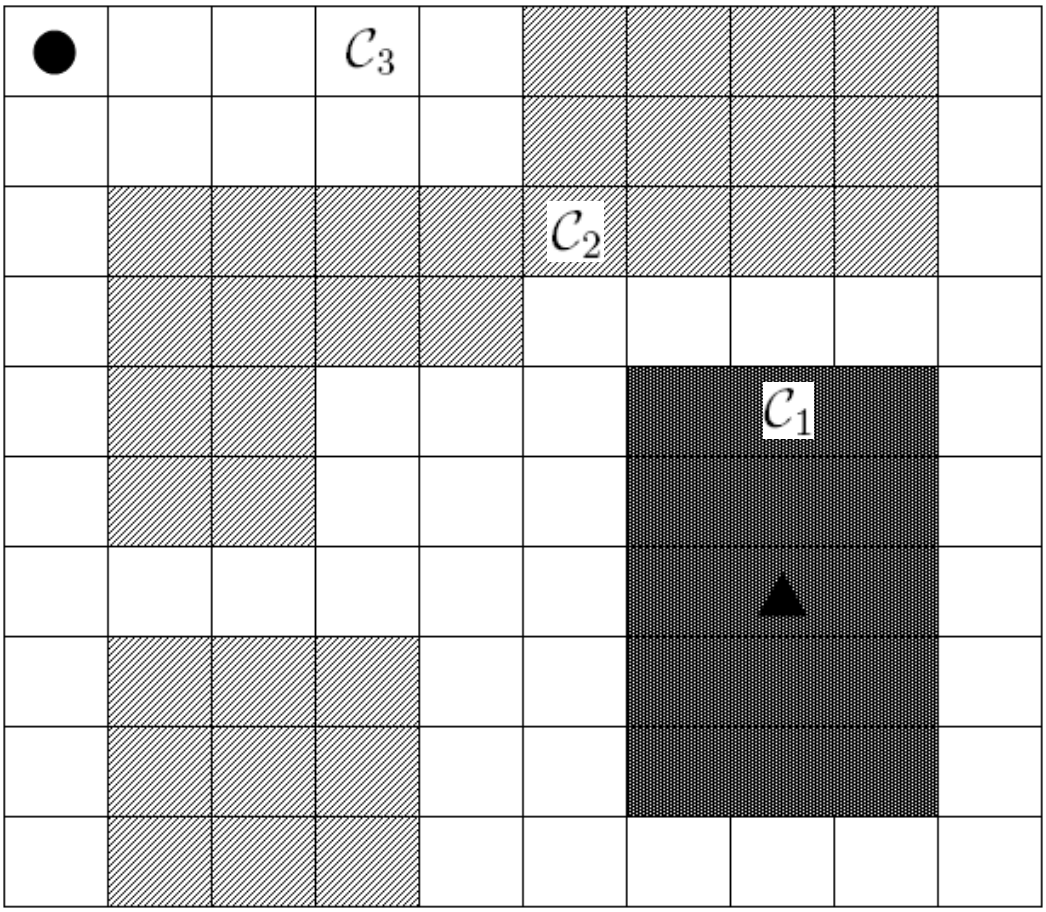

Figure 6: An area of interest composing of 10 by 10 cells and two altitudes. Heavily-shaded cells ($\mathcal{C}_1$), lightly-shaded cells ($\mathcal{C}_2$), and unshaded cells ($\mathcal{C}_3$) describe risky, moderately risky, and non-risky area respectively. The circle indicates the cell over which searcher starts and the triangle specifies the initial position of the target.

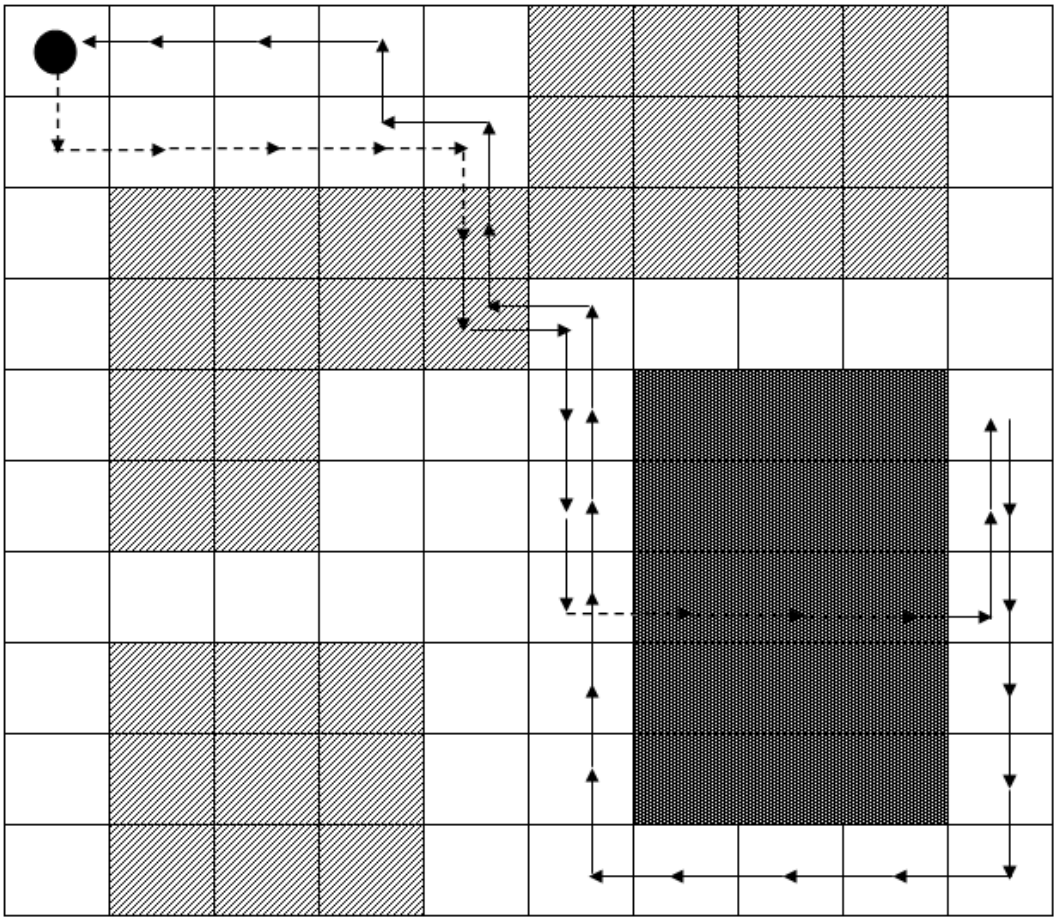

Figure 7: An optimal path for survival probability limit 0.90 and fuel limit 400. The solid lines and the dashed lines represent flight segments at low and high altitude, respectively.

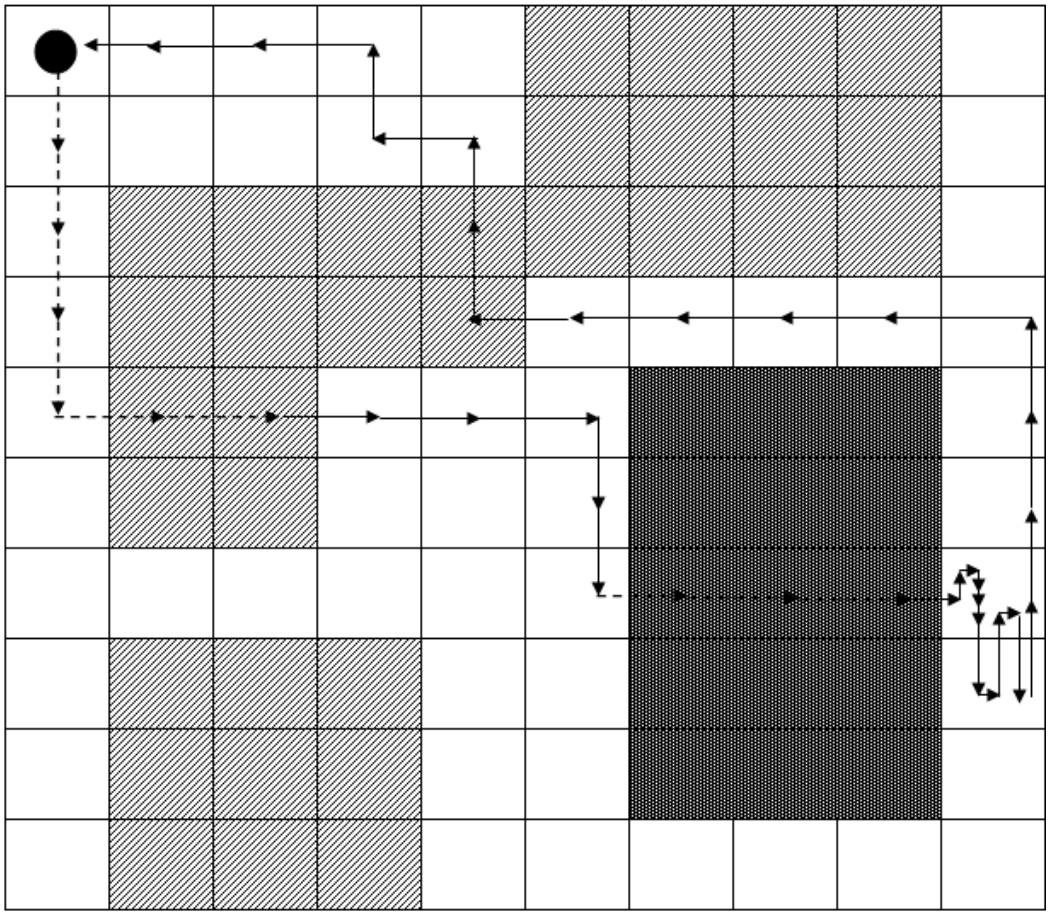

Figure 8: An optimal path for a case with edge-dependent glimpse probability, survival probability limit 0.90, and fuel limit 400. The solid lines and the dashed lines represent flight segments at low and high altitude, respectively.